НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Веб-сервіс для укладання безпечних угод оренди без посередників на основі технології Blockchain»

Виконала:
студентка IV курсу, групи КП-61
Чумак Надія Русланівна _____

Керівник:
Старший викладач кафедри ПЗКС,
Гадиняк Руслан Анатолійович _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
Старший викладач кафедри ПМА,
Мальчиков Володимир Вікторович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

«ЗАТВЕРДЖУЮ»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___»_____2019 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Чумак Надії Русланівни

1. Тема проєкту «Веб-сервіс для укладання безпечних угод оренди без посередників на основі технології Blockchain», керівник проєкту Гадиняк Руслан Анатолійович, старший викладач, затверджені наказом по університету від «25» травня 2020 р. № 1181-с

2. Термін подання студентом проєкту: «15» червня 2020 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - аналіз існуючих рішень;
   - аналіз мов програмування, технологій розроблення;
   - огляд розроблених програмних засобів;
   - аналіз реалізації програмного забезпечення.

5. Перелік обов'язкового ілюстративного матеріалу:
   - функціональність програмних засобів (креслення);
   - структура бази даних (креслення);
   - схема алгоритму роботи смарт-контракту (плакат);
   - схема взаємодії програмних модулів (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	16.11.2019	
2.	Розробка та узгодження технічного завдання	25.11.2019	
3.	Підготовка матеріалів першого розділу дипломного проєкту	25.12.2019	
4.	Розроблення структури веб-сервісу	16.01.2020	
5.	Підготовка матеріалів другого розділу дипломного проєкту	01.03.2020	
6.	Розроблення смарт-контракту для укладання угод	22.03.2020	
7.	Програмна реалізація і тестування веб-сервісу	12.04.2020	
9.	Підготовка третього розділу дипломного проєкту	25.04.2020	
10.	Підготовка четвертого розділу дипломного проєкту	02.05.2020	
11.	Підготовка графічної частини дипломного проєкту	19.05.2020	
12.	Оформлення документації дипломного проєкту	25.05.2020	

Студентка Надія ЧУМАК

Керівник проєкту Руслан ГАДИНЯК

АНОТАЦІЯ

Основною темою даного дипломного проєкту є програмна реалізація веб-сервісу укладення договорів оренди без посередників з використанням розумних контрактів та технології блокчейн.

Проаналізовано аналоги в двох сферах – децентралізовані та централізовані сервіси та платформи, що надають послуги оренди нерухомості на різні терміни. Були враховані основні вимоги до сервісу, як продуктові (нефункціональні), так і користувацькі (функціональні), та виокремлені найбільш важливі для імплементації функції.

Платформа представляє собою веб-сервіс, розроблений згідно останніх трендів веб-сфери інформаційних технологій, таких як динамічне оновлення сторінок (SPA), відповідність міжнародним стандартам збереження особистої інформації користувачів (GRPR) та інші. Система динамічно реагує на роль, рівень та наявність відкритої сесії і показує різний веб-контент в залежності від цих факторів. Деяка функціональність доступна лише авторизованим користувачам, а доступ до адміністративної панелі є лише у користувачів з адмін правами. Серверна частина підтримує хешування особистих даних користувачів згідно алгоритму “bcrypt”, а також використовує окреме надійне сховище для зберігання паролів від гаманців користувачів на платформі (Vault Encrypted Storage). Було розроблено власний смарт-контракт, написаний мовою Solidity, відповідна копія якого в мережі блокчейн Ethereum супроводжує кожну угоду укладену в межах сервісу.

Даний дипломний проєкт включає в себе: модуль управління інформацією користувачів, модуль укладення угод, модуль взаємодії з смарт-контрактом та мережею блокчейн, модуль інформації орендодавців, модуль роботи з базою даних та захищеними сховищами даних, адміністративний модуль, модуль KYC, а також візуальне (графічне) супроводження у вигляді адаптивного дизайну веб сторінок.

ABSTRACT

The main theme of the diploma project is the software implementation of a web service for lease agreements without intermediaries using smart contracts and blockchain technology.

Analogs in two spheres have been analyzed – decentralized and centralized services and platforms that provide rental services for different time periods. The basic requirements for the service, both product (non-functional) and user (functional) were taken into account, and the most important functions for implementation were singled out.

The platform itself is a web service developed according to the latest trends in the web part of information technology, such as dynamic page content fetch (SPA), compliance with international standards for the general data protection regulation (GRPR) and others. The system responds dynamically to the role, level, and availability of an open session and displays different web content based on these factors. Some functionality is available only to authorized users, and access to the admin panel is available only to users with admin rights. The server part supports hashing of personal data of users according to the "bcrypt" algorithm, and also uses separate safe storage for storing passwords from user's platform wallets (Vault Encrypted Storage). Service implements our own smart contract, written in the Solidity language, a corresponding copy of which in the Ethereum blockchain network accompanies each agreement made within the service.

This diploma project includes the module of user information management, the module of rental agreements, the module of interaction with smart contract and blockchain network, the module of landlords information, the module of database and secure data warehouses, administrative module, module KYC, and visual (graphic) support in the form of adaptive design of web pages.

ДП.045440-01-90 Веб-сервіс для укладання безпечних угод оренди без посередників на основі технології Blockchain. Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Веб-сервіс для	5	
	укладання безпечних		
	угод оренди без		
	посередників на основі		
	технології Blockchain.		
	Технічне завдання		
ДП.045440-03-81	Веб-сервіс для	62	
	укладання безпечних		
	угод оренди без		
	посередників на основі		
	технології Blockchain.		
	Пояснювальна записка		
ДП.045440-04-51	Веб-сервіс для	4	
	укладання безпечних		
	угод оренди без		
	посередників на основі		
	технології Blockchain.		
	Програма та методика		
	тестування		
ДП.045440-05-34	Веб-сервіс для	17	
	укладання безпечних		
	угод оренди без		
	посередників на основі		
	технології Blockchain.		
	Керівництво користувача		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-06-99	Веб-сервіс для	1	
	укладання безпечних		
	угод оренди без		
	посередників на основі		
	технології Blockchain.		
	Функціональність		
	програмних засобів. UML		
	діаграма прецедентів		
ДП.045440-07-99	Веб-сервіс для	1	
	укладання безпечних		
	угод оренди без		
	посередників на основі		
	технології Blockchain.		
	Структура бази даних.		
	ER діаграма		
ДП.045440-08-98	Веб-сервіс для	1	
	укладання безпечних		
	угод оренди без		
	посередників на основі		
	технології Blockchain.		
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2019 р.

ВЕБ-СЕРВІС ДЛЯ УКЛАДАННЯ БЕЗПЕЧНИХ УГОД ОРЕНДИ НА ОСНОВІ ТЕХНОЛОГІЇ BLOCKCHAIN

Технічне завдання

ДП.045440-02-91

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Руслан ГАДИНЯК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Надія ЧУМАК

2019

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: веб-сервіс для укладання безпечних угод оренди без посередників на основі технології Blockchain.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для укладання безпечних угод із збереженням автономності та конфіденційності інформації, а також з метою зменшення вартості, оскільки цифровий алгоритм обходиться значно дешевше, ніж реальний юридичний процес.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Програмна система повинна забезпечувати такі основні функції:

1. Можливість перегляду списку усіх доступних варіантів оренди.
2. Вивід та збереження інформації про укладені угоди.
3. Створення приватного ключа і цифрового підпису як гаранту надійності.
4. Можливість оплати оренди в криптовалютах.
5. Переказ оплати за актуальним курсом валюти по вказаній адресі смарт-контракту, після підтвердження бронювання.
6. Введення паспортного номеру як паролю до розблокування коштів. Паспортний номер буде захешований за допомогою

алгоритму SHA-256 відразу після введення, номер паспорту не показується в явному вигляді і буде недоступний для перегляду.

7. Можливість отримання оплати тільки після фізичної присутності орендаря та наявності паспорта, який слугує паролем для розблокування коштів.

Розроблення серверної частини виконати на мові програмування Ruby, використовуючи веб-фреймворк Ruby on Rails. Розроблення смарт-контракту для укладання угод без посередників виконати на мові програмування Solidity. Розроблення клієнтської частини виконати, використовуючи веб-фреймворк React для забезпечення інтерактивності та динамічності веб-сторінки.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

1. Пояснювальна записка.
2. Програма та методика тестування.
3. Керівництво користувача.
4. Креслення:
   - «Структура бази даних. ER діаграма»;
   - «Функціональність програмних засобів. UML діаграма прецедентів».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою проєкту……...16.11.2019

Розроблення та узгодження технічного завдання……25.11.2019

Підготовка матеріалів першого розділу дипломного проєкту…….25.12.2019

Розроблення структури веб-сервісу……..16.01.2020

Підготовка матеріалів другого розділу дипломного проєкту……01.03.2020

Розроблення смарт-контракту для укладання угод……..22.03.2020

Програмна реалізація і тестування веб-сервісу……12.04.2020

Підготовка третього розділу дипломного проєкту…….25.04.2020

Підготовка четвертого розділу дипломного проєкту……02.05.2020

Підготовка графічної частини дипломного проєкту……19.05.2020

Оформлення документації дипломного проєкту…….25.05.2020

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до «Програми та методики тестування».

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

ВЕБ-СЕРВІС ДЛЯ УКЛАДАННЯ БЕЗПЕЧНИХ УГОД ОРЕНДИ БЕЗ ПОСЕРЕДНИКІВ НА ОСНОВІ ТЕХНОЛОГІЇ BLOCKCHAIN

Пояснювальна записка

ДП.045440-03-81

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Руслан ГАДИНЯК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Надія ЧУМАК

2020

ЗМІСТ

ВСТУП

Найбільшим винаходом останніх років стала технологія блокчейн. Саме розподілена база даних блокчейн дала змогу реалізувати концепт криптовалют, які вже назвали валютами майбутнього. Основою ідеєю технології є зберігання даних, захищених від підробки та спотворення, а отже, і від більшості хакерських атак. Зазвичай дані наявні у відкритому доступі по хешу відповідного блока всередині мережі блокчейн. Особливості технології відкрили багато нових шляхів для її інтегрування у різних сферах життя суспільства, де особливе значення має питання безпеки та децентралізованості.

Блокчейн – це технологія, за допомогою якої можна реформувати та забезпечити прозорість та децентралізованість. Не дивно, що найбільше вона вплинула на ринок торгівлі. Окремою нішею є ринок нерухомості, де на сьогоднішній день існуючі сервіси використовують систему, при якій гарантом угоди виступає сама платформа (компанія-посередник), так як лише вона може підтвердити права та обов'язки сторін. За цю роль посередник бере комісійні, які часто досягають 10-15% від номінальної вартості угоди. За допомогою технології блокчейн можна мінімізувати людські ресурси (виключити сторону-посередника) і таким чином зменшити комісійні витрати, адже сама технологія гарантує надійність та достовірність усього, що містить база даних.

Таким чином, даний дипломний проєкт присвячено реалізації вищеописаного концепту в доступній для звичайного користувача формі – у вигляді веб-сервісу для укладання угод, в основі яких покладено вирішення основних проблем існуючої паперової системи шляхом використання технології блокчейн та смарт-контрактів.

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Back end – частина комп'ютерної системи або програми, до якої безпосередньо не звертається користувач. Зазвичай відповідає за зберігання та маніпулювання даними.

Front end – це інтерфейс для взаємодії між користувачем і back end.

API, Application Programming Interface – у програмному забезпеченні, набір використовуваних функцій, інтерфейс для створення програмних додатків.

Блокчейн – розподілена база даних, що зберігає впорядкований ланцюжок записів (так званих блоків), що постійно довшає.

Смарт-контракт, Smart contract – різновид угоди в формі закодованих математичних алгоритмів, укладення, зміна, виконання і розірвання яких можливо лише з використанням комп'ютерних програм (блокчейн платформ) в рамках мережі Інтернет.

Криптовалюта – цифрова, віртуальна валюта, що використовує криптографічні алгоритми асиметричного шифрування для забезпечення безпеки транзакцій. Криптовалюту також називають криптомонетою.

Ethereum – криптовалюта та платформа для створення децентралізованих онлайн-сервісів на базі блокчейна, що працюють на базі розумних контрактів.

Газ (Gas) – це одиниця обчислення, яка позначає розмір комісії за певну дію або транзакцію.

Лімітгазу (Gas Limit) – це максимальна кількість газу, яке користувач готовий заплатити за виконання цієї дії або підтвердження транзакції (мінімум - 21,000).

Ціна газу (Gas Price) – це кількість Gwei (найменша неподільна одиниця ETH), яке користувач готовий витратити на кожну одиницю газу.

Альткоін – будь-яка цифрова криптовалюта, відмінна від Bitcoin. Термін походить від «альтернатива до Bitcoin» та використовується для описання криптовалюти, що не є Bitcoin.

Фіатні гроші – тип грошей або валюти, цінність яких походить не від власної вартості або гарантії обміну на золото або іншу валюту, а від державного наказу використання їх як засобу платежу.

ICO, Initial Coin Offering – одна із форм колективного фінансування, краудфандингу, спосіб залучення фінансових інвестицій у проєкт. При такому способі збору інвестицій випускається валюта, яка розподіляється серед зацікавлених осіб.

Airdrop – це процес роздачі токенів користувачам блокчейн-гаманців на безоплатній основі. Цей метод зазвичай використовується стартап проєктами, які проводять ICO в якості інструменту для просування проєкту і підвищення впізнаваності бренду.

RESTful API – API, що використовує HTTP запити GET, PUT, POST та DELETE для забезпечення доступу до інформації всередині системи стороннім користувачам.

Алгоритм SHA-256, Secure Hash Algorithm – односпрямована функція для створення цифрових відбитків фіксованої довжини (256 біт, 32 байт) з вхідних даних розміром до 2,31 ексабайт (2^{64} біт).

DOM, Document Object Model – інтерфейс, що дозволяє програмам та скриптам динамічно отримувати доступ і оновлювати вміст, структуру та стиль html документа.

Ajax, Asynchronous JavaScript and XML – технологія для побудови інтерактивних веб-додатків, яка оброблює запити користувача одразу при їх приході.

KYC, Know Your Customer – процедура для верифікації та ідентифікації особистості клієнта, який має намір провести будь-яку фінансову операцію. KYC можна назвати частиною процедур, спрямованих

на запобігання незаконного відмивання грошей, допомагає організаціям уникнути шахрайства.

JSON, JavaScript Object Notation – один із форматів текстового обміну даними, є легко зчитуємим для людини, використовується для представленя структур даних (масив, словник тощо) у текстовому вигляді.

Grape – JSON-API фреймворк для Ruby. Він має багато схожого з Rails. Він чітко визначений, але не нав'язує жорсткої структури та відповідає лише за маршрутизацію та обробку даних.

Material-UI – ReactJS's фреймворк, що надає готові google рішення для швидкої і досить простий веб-розробки. Material-UI досить велика бібліотека, де ключовою частиною react компонентів і стилізації є @material-ui/core.

Модель-Вид-Контролер, MVC – архітектурний шаблон, який використовується під час проєктування та розробки програмного забезпечення.

JWT, JSON Web Token – веб-токен, призначений для передачі підписаних "заявок" між службами (як зовнішніми, так і внутрішніми для вашого застосунку/сайту). "Заявки" – частина інформації, яку інші можуть переглядати та / або перевіряти, але не змінювати.

Solidity – це мова програмування для розробки смарт-контрактів на Ethereum.

Cookies – невеликі текстові або бінарні дані, отримані від веб-сайту на веб-сервері, які зберігаються у клієнта, тобто браузера, а потім відправляються на той самий сайт, якщо його буде повторно відвідано.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1. Аналіз особливостей укладення угод оренди без посередників

На сьогодні договори оренди вважаються одними із найбільш використовуваних у суспільстві. Як і інші типи договорів, він визначає обов'язки та права для двох сторін (орендар та орендодавець в конкретному випадку), якими можуть виступати підприємства та підприємці (юридичні особи), а також фізичні особи [1]. Дуже важливим аспектом є те, що такі договори повинні містити чіткий опис майна, яке надаватиметься в тимчасове користування, та правила користування. Якщо умови будуть порушені, чи предмет оренди отримає пошкодження, непередбачені договором (фізичне зношення предмету активного використання – лижі, одяг, спорт. інвентар тощо), то орендар має понести відповідну, вказану в договорі кару. Зазвичай орендар має виплатити компенсацію або за власний кошт виконати ремонтні чи косметичні роботи, якщо відмінне не вказано в договорі. Також існує цілий ряд правових чинників, за яких угода може вважатися недійсною, бути розірвана раніше домовленого строку тощо [2]. Зрозуміло, що такі тонкощі питання невідомі звичайній пересічній людині, а отже, зупинимось на особливостях оренди детальніше.

Наведемо основні відмінності договорів оренди та інших договорів:

1. Майно передається у користування лише на узгоджений строк.
2. Майно має бути обов'язково повернуто орендодавцю не пізніше терміну завершення терміну дії договору або за бажанням однієї чи двох сторін (дострокове розірвання), в іншому випадку можливе додаткове стягнення коштів.
3. Внесення плати за використання орендованого майна та обслуговування, пов'язане з цим – обов'язкова умова.

Основними проблемами укладання угод є:

1. Неможливість укладання угоди без посередника (юридична безграмотність людей).
2. Ризики укладання угод в усній формі на “довірі”.
3. Відсутність важливої інформації в публічному доступі про “репутацію” орендаря та орендодавця.
4. Відсутність єдиного світового регулятора (підпадання інтернаціональних угод під закони двох держав).
5. Витрата часу. Велика кількість бюрократичних процедур вимагає постійного завірення та підписання документів, а також їх зберігання, що потребує безпосередньої фізичної присутності як однієї, так і другої сторони.

Вищеописані проблеми не мають вирішення вже багато років, але з винаходом технології блокчейн все змінилось. Блокчейн можна успішно інтегрувати в сферу договорів оренди, технологія пропонує ряд важливих особливостей [3]:

1. Децентралізованість – відсутність стороннього зацікавленого впливу та контролю, який дозволяє системі розвиватись самостійно, без перешкод.
2. Стабільність – інформація зберігається всередині мережі, а отже, втрата даних фізично та теоретично не можлива, незалежно від обставин.
3. Захищеність – в основі реалізації технології лежить взаємопідтвердження блока інформації іншими блоками та користувачами мережі, тому можливість підміни стає мінімальною.
4. Незмінність – для того, аби внести зміни в інформацію, збережену в будь-якому блоці мережі, треба внести зміни і до всіх наступних блоків мережів. Для цього потрібна така ж потужність, яка створила всі наступні блоки, тому можливість

внесення змін в будь-який блок стає мінімальною (фактично такою, яку на практиці реалізувати неможливо).

1.2. Огляд існуючих сервісів для укладання угод без посередників

Під час аналізу існуючих систем програмних рішень, а саме веб-сервісів, що використовують технологію блокчейн у сфері нерухомості, було знайдено кілька працюючих прототипів. Усі проєкти мають базу активних користувачів, а отже, користуються популярністю в межах даної ніші ринку. Більшість вищеописаних проєктів функціонують вже декілька років, що свідчить про попит та постійний інтерес до використання технології блокчейн у сфері торгівлі (оренди) нерухомості.

Веб-сервіс Rentberry є одним із яскравих представників інноваторів у вищеописаній сфері [4]. Платформа позиціонує себе як децентралізована, що автоматизує основні паперові (як бюрократичні, так і фінансові) процедури для надання швидкого доступу до послуг купівлі/оренди нерухомості. Rentberry було засновано в 2015 році. Це перша компанія, що почала використовувати блокчейн технологію в сегменті довгострокової оренди. До появи Rentberry блокчейн аналогів старій паперово-бюрократичній системі не було.

Основною заявленою перевагою платформи є відсутність агента або брокера, а отже, і відсутність усіх супутніх витрат, а також повний доступ та можливість вести справи орендарям та орендодавцям у режимі онлайн. Rentberry пропонує стандартний перелік юридичних послуг та повноцінний завірений контракт. Перевагою сервісу є те, що підписати контракт можна електронно, так як партнером платформи є HelloSign, який надає послуги електронних підписань, що значно заощаджує час користувачу та платформі. Також Rentberry підтримує завантаження та оцифровування “паперових” контрактів. Платформа надає місце для зберігання електронної копії договору, що зберігається з відповідним рівнем шифрування та безпеки.

Концепт Rentberry наполягає на доцільності створення єдиного стандарту процедур оренди та купівлі-продажу, що значно полегшує процес для експантів, що знаходяться за кордоном, а також відкриває нові шляхи для іноземних інвестицій.

Веб-сервіс Rentberry виділяється довгим послужним списком серед інших конкурентів. На 2019 рік система нараховує вже близько 120 тисяч користувачів та близько 240 тисяч об’єктів (рис. 1.1). Сервіс охоплює всю територію США, але в найближчих планах у платформи є вихід на європейський та азіатський ринок, адже велика частина інвесторів саме з Китаю, Європи та Близького Сходу.

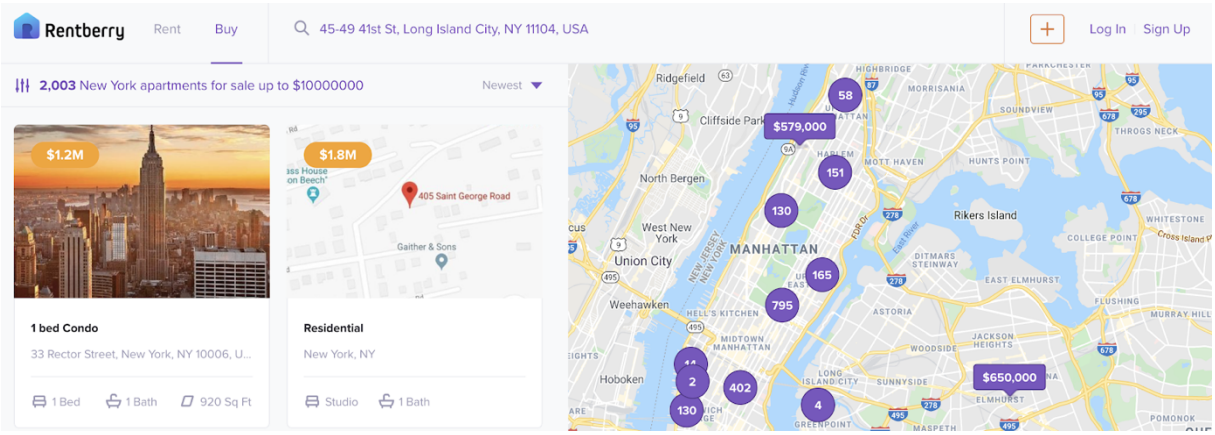


Рис. 1.1. Вибір об’єкту для купівлі на території США у веб-сервісі Rentberry

Веб-сервіс розраховує полегшити життя як орендодавцям, так і орендарям. Одним із нововведень платформи та її особливістю є аукціонний режим оренди. Такий режим найбільш популярний, у так звані “високі” сезони, або в місцях підвищеної популярності. Це дозволяє користувачам прийняти участь у бронюванні навіть високопопулярних об’єктів, а орендодавцям отримати гідну оплату за надані послуги. З іншого боку, в деяких випадках аукціонний режим дозволить орендувати житло за ціною нижче заявленої, а орендодавцю дасть можливість не залишати об’єкт без торгового обороту. На рис. 1.2 зображено форму ставки в аукціонному

режимі. Для участі в аукціоні користувачу потрібно мати 1000 BERRY токенів. У разі виграшу в аукціоні 950 токенів йдуть в рахунок оплати застави на період оренди, а 50 токенів відходять платформі в якості комісії, саме так платформа отримує зиск з функціонування та надання послуг. Незважаючи на присутність комісійних, їх кількість залишається в сотні разів менше, ніж аналогічні традиційні комісійні.

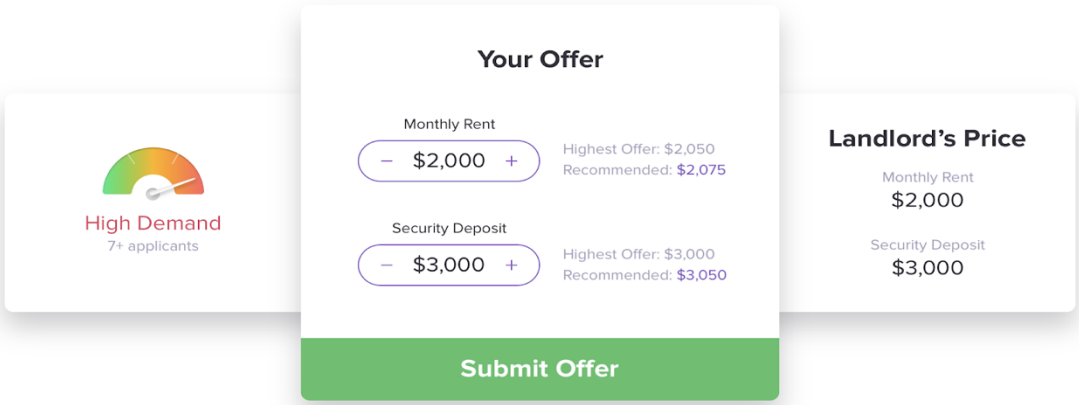


Рис. 1.2. Аукціонний режим веб-сервісу Rentberry

Однією з особливостей сервісу Rentberry є використання власного токена BERRY (згаданого вище), як основного та єдиного методу оплати всіх послуг на платформі. Токен має чітку ціну в фіатних валютах та може бути обміняний на євро, доллар. BERRY знаходиться на таких платформах-обмінниках як IDEX, HitBTC. Основна ідея токена – це стерти кордони між користувачами платформи в незалежності від їх внутрішніх регуляторів. ICO також стало одним із основних інвестиційних доходів для компанії та двигуном прогресу на перші два роки розвитку компанії. Також, на базі технології блокчейн та реалізованому з її допомогою токенів BERRY сервіс розробив соціальні програми. Користувачі можуть допомагати один одному з оплатою застави на період оренди та отримуватимуть за це винагороду у вигляді 3-7% токенів.

Веб-сервіс Bitsrent є другою по величині після Rentberry платформою зі схожою функціональністю, який також поклав в основу своєї роботи технологію блокчейн [5]. На відміну від попереднього представника, сервіс не мав великих інвестицій, а всі кошти на розробку та запуск платформи були отримані в ході ICO токена власної розробки BTRS. Цікаво те, що проєкт підтримали як крипто-ентузіасти, так і звичайні користувачі, що яскраво показує популярність даного рішення в масах.

Bitsrent грамотно розподілив кошти отримані з ICO для правильного та успішного розвитку компанії. Детальніше про розподілення інвестицій на рис. 1.3.

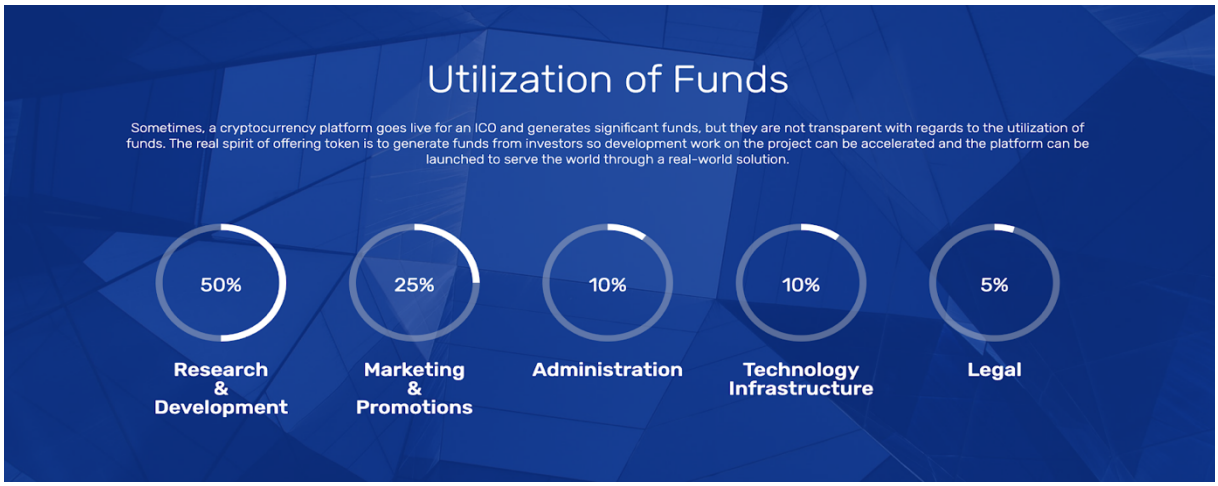


Рис. 1.3. Розподілення коштів отриманих в ході ICO Bitsrent

За допомогою стандартних для цієї сфери методів маркетингу таких, як air drop та interest payments, було набрано початкову базу користувачів. Веб-сервіс Bitsrent має як цілий ряд відмінностей від інших платформ, так і низку схожих рішень. Основні завдання, які ставить перед собою сервіс можна виділити в декілька тез.

Bitsrent робить великий акцент на прозорості технології блокчейн, вважаючи, що стандартна централізована модель не реалізує ніяких методів захисту від шахрайської діяльності і відслідкувати фактичного володаря нерухомості або об’єкту оренди іноді неможливо. У варіанті Bitsrent

використовується Orbit Blockchain та мережа Ethereum. База даних блокчейн зберігає всю інформацію про угоди між сторонами, платіжну та реквізитну інформацію. А смарт-контракти виступають своєрідним регулятором відносин та гарантом непорушності угоди та її умов, що знімає всі питання щодо правдивості сторін.

Другою проблемою, вирішення якої ставить перед собою Bitsrent, є неточність записів клієнтів та виставлення рахунків. Дуже часто внаслідок стихійних лих втрачаються важливі оригінали угод, документів, що підтверджують право власності. Оскільки відновлення часто вимагає великого проміжку часу, то це призводить до неточностей в оплаті та виставлених рахунках. Технологія блокчейн гарантує незмінність даних, що зберігаються децентралізовано, а тому мають безліч копій. Дані постійно оновлюються з кожним новим блоком, новою транзакцією, а отже, втрати даних унеможливлюються.

Смарт-контракти, у свою чергу, дозволяють гарантувати оплату по угоді без будь-яких додаткових чи не обговорених комісій. Bitsrent як онлайн-платформа гарантує безпечне збереження приватної інформації користувачів. Orbit Blockchain реалізує такі методи захисту, які не можуть бути зламані стандартними методами кібер-атак, чи DDOS, а відсутність фізичних носіїв з даними гарантує неможливість їх викрадення та зловживання даними (від чого часто страждає звичайна централізована система).

Токен BTR заслуговує окремого опису. Особливістю його є те, що він використовується не тільки, як внутрішня валюта оплати, а і як частина спільноти. Більшість токенів у власності простих користувачів, певна частина зарезервована для ентузіастів та інвесторів, для різних конкурсів, і лише невелика частина для команди, що підтверджує направленість розробки саме на спільноту. Також в планах команди є знищення певної кількості токенів для підтримання їх цінності. Більш детальну інформацію про розподілення токенів на рис. 1.4.

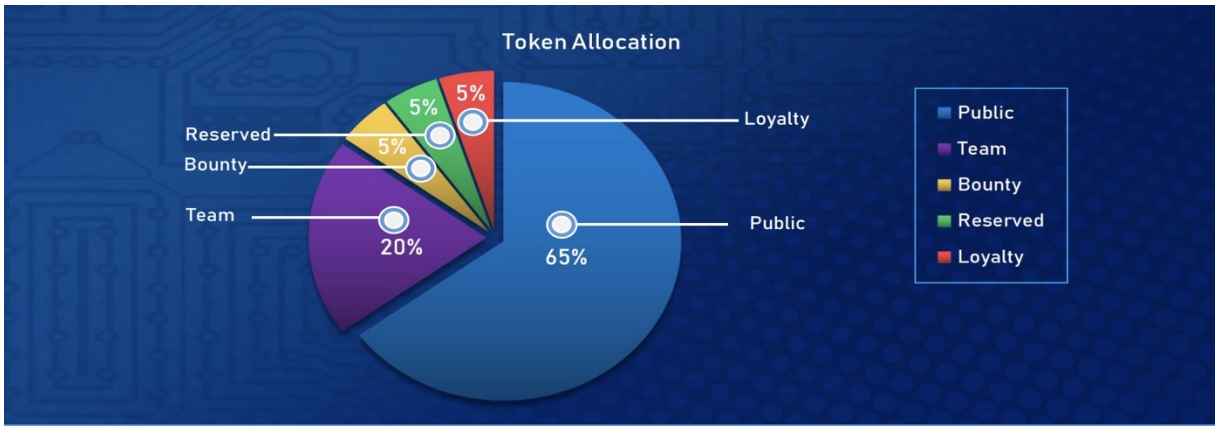


Рис. 1.4. Розподілення токенів BTR сервісу Bitsrent

Веб-сервіс Staybit – це децентралізована платформа, що надає послуги короткострокової оренди та орієнтована на людей, що шукають оселю під час відпусток чи закордонних поїздок [6]. Основну увагу система приділяє смарт-контрактам, щоб в автоматичному режимі вирішувати більшість поширених конфліктів, а також безпечно переказувати кошти між орендарями та орендодавцями. Найбільшу проблему у сучасній централізованій системі Staybit вбачає саме в наявності посередника і регулятора. Через участь першого з'являються комісійні, які часто досягають 20% від вартості оренди (airbnb, booking [7]), а наявність другого значно ускладнює процес оплати, адже платіж може бути відхилений або заблокований фінансовою установою та має відповідати законам обох сторін. Завдяки цілому комплексу заходів таких, як децентралізація, зміна фіатних валют на криптовалюту, введення смарт-контрактів при депозиті та оплаті, що виконуються в межах технології блокчейн, система Staybit пропонує користувач ряд переваг – значно зменшена вартість транзакцій (в 2, 3 рази менше від сучасних банківських), більше свободи у виборі для користувача, незалежно від законів регулятора та фінансово-економічних відносин між країнами сторін, захист користувачів від шахрайства та обману, спрощення закордонних платежів, як виду сплати. На рис. 1.5 можна побачити приклади декількох сторінок меню сервісу

Staybit. Одразу помітно, що інтерфейс не акцентує увагу користувача на крипто- та блокчейн сфері.

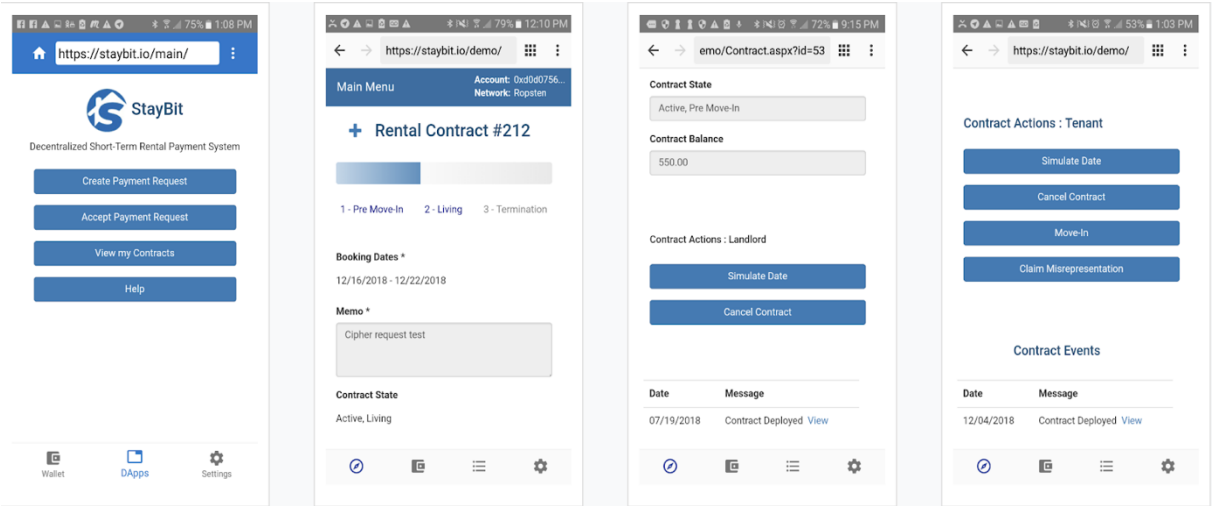


Рис. 1.5. Загальне меню та сторінки сервісу Staybit

На відміну від більшості сервісів, що мають в своїй основі технологію блокчейн, Staybit намагається приховати та вберегти кінцевого користувача від всіх тонкощів роботи з криптовалютами (зберігання альткоіну ERC-20, gas price, комісійні за транзакції мережі блокчейн тощо). Задля цією мети інтерфейс сервісу максимально нагадує стандартні, централізовані аналоги.

Як і інші конкуренти, Staybit має власний токен, але відмінність полягає в тому, що сервіс Staybit не пропонує функціональності зберігання валюти на платформі, а тому для роботи з ним потрібен браузер з підтримкою Web3 або браузери Trust, Coinbase Wallet (Toshi) або Cipher, в тому числі і для мобільних платформ iOS або Android. Але це не має великого впливу на інтерфейс, а отже, не приносить ніяких додаткових складностей у використанні платформи користувачем (рис. 1.6). Staybit реалізує договори оренди трьох типів (гнучкий, стандартний, суворий). Така система дозволяє імітувати роботу відомих платформ AirBnb і VRBO.

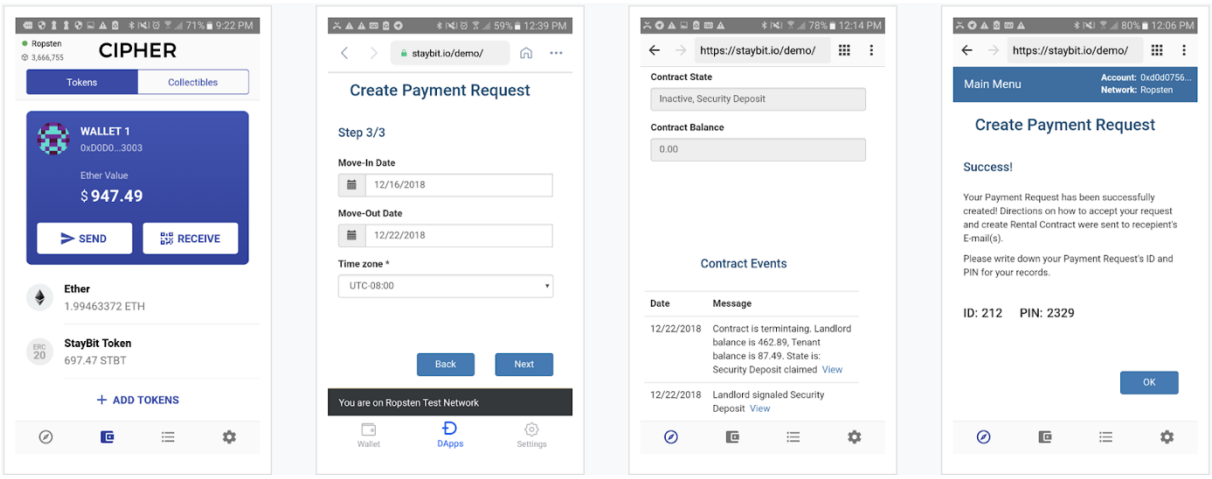


Рис. 1.6. Інтерфейс гаманця та грошових переведень платформи Staybit

1.3. Аналіз вимог до функціональності програмних засобів

Згідно отриманих в ході аналізу даних, можна стверджувати, що описані в попередньому розділі сервіси не пропонують в повній мірі функціональності, яка б вирішувала основні проблеми сучасної системи оренди нерухомості [8] та задовольняла потреби пересічного користувача. Кожна із платформ виокремлює лише частину загальної проблеми, жертвуючи при цьому іншими її, не менш важливими, компонентами. Маючи багато спільного (використання технології блокчейн, смарт-контрактів, робота з орендодавцями та орендарями напряму) ні один з наведених сервісів не може в повній мірі конкурувати з визнаними та популярними на сьогоднішній день централізованими платформами надання послуг оренди, адже не надають еквівалентно достатніх умов для комфортного користування.

Відповідно до результатів було виокремлено порівняльну таблицю результатів аналізу програмних рішень (табл. 1.7).

Отже, для повноцінного покриття проблеми надання послуг оренди нерухомості було визначено низку вимог:

1. Платформа повинна слідувати політиці відсутності ролі посередника в наданні послуг та комісій пов’язаних з цим.

2. Сервіс повинен підтримувати надання послуг різної тривалості (короткострокова та довгострокова оренда).
3. Платформа має гарантувати низький поріг входу для користувача (можливість легко придбати валюту розрахунку платформи – токен).
4. Сервіс повинен надавати юридичні підтвердження укладених контрактів.
5. Платформа повинна бути гарантом угоди та відшкодовувати / повертати кошти в разі будь-яких проблем чи невідповідностей.

Таблиця 1.7

Порівняння результатів аналізу програмних рішень

	Критерії				
Програмні рішення	Безкомісійне з'єднання	Оренда будь-якої тривалості	Зрозумілий інтерфейс та процес користування	Юридично правильні угоди	Гарантія повернення коштів
Rentberry	+/–	+/–	+/–	+	+/–
Bitsrent	+	–	+/–	+	–
Staybit	+	+	+	+/–	+/–

Додатково, програмне рішення повинно усунути такі проблеми:

1. Швидкість заключення угоди та проведення оплати.
2. “Свобода” в міждержавних платежах та відсутність комісій пов’язаних з державними регуляторами.
3. Захист особистої інформації та достовірність наданої інформації про нерухомість.

4. Відсутність способів обману та махінацій з грошима, захищеність користувачів та можливість блокування скомпрометованих користувачів.
5. Захищеність важливих документів та інформації щодо угод від сторонніх чинників, що можуть призвести до часткової або повної втрати даних, а отже, і до порушення норм угоди.

На основі результатів, отриманих від порівняльного аналізу існуючих рішень та вимог до проєкту, маємо такі сформовані завдання для розробки програмного застосунку:

1. Можливість перегляду списку усіх доступних варіантів оренди (як довгострокової, так і короткострокової).
2. Оплата у валюті, яку легко можна купити та продати за фіатні кошти або обміняти на інші види криптовалют.
3. Відсутність комісійних за користування платформою та реєстрацію нерухомості.
4. Вивід та збереження інформації про укладені угоди. Забезпечення стабільної роботи та доступу до інформації користувачу або довіреній особі.
5. Створення приватного ключа і цифрового підпису як гаранту надійності та непорушності угоди. Зберігання інформації сторін для забезпечення дотримання прав та обов’язків сторін.
6. Можливість оплати оренди в криптовалютах, незалежно від місця проживання користувача та країни місцезнаходження нерухомості, що фігурує в угоді.
7. Переказ оплати за актуальним курсом валюти по вказаній адресі смарт-контракту, після підтвердження бронювання та неможливість стягнення коштів без підтвердження виконання угоди.
8. Введення паспортного номеру як паролю до розблокування коштів. Паспортний номер буде захешований за допомогою

алгоритму SHA-256 відразу після введення, номер паспорту не показується в явному вигляді і буде недоступний для перегляду. Забезпечення захищеного методу зберігання особистої інформації, включаючи паспортні дані для того, аби дані не були скомпроментовані.

9. Можливість отримання оплати тільки після фізичної присутності орендаря та наявності паспорта, який слугує паролем для розблокування коштів. Гарантом угоди та захистом для махінацій має виступати непорушна та незмінна автоматична система (смарт-контракт).

2. АНАЛІЗ МОВ ПРОГРАМУВАННЯ, ТЕХНОЛОГІЙ РОЗРОБЛЕННЯ

2.1. Вибір веб-сервісу в якості типу програмного забезпечення

Веб-сервіс – це програма, яка дозволяє обмінюватись інформацією між комп'ютерами через мережу. Веб-сервіси зазвичай представляють собою клієнт-серверну архітектуру, в основу якої покладено обслуговування клієнтів за рахунок розміщення значної частини ресурсів на серверах.

Можна виділити низку переваг веб-сервісу [9], а саме:

- не потребує встановлення на персональний комп'ютер;
- доступ до даних незалежно від операційної системи;
- простота налаштування за рахунок відкритих стандартних протоколів;
- простий в розгортанні та інтеграції;
- можливе одночасне використання декількох версій сервісів;
- незалежність клієнтського та серверного коду;
- стійкість до одночасного використання сервісу великою кількістю користувачів;
- відстеження дій користувачів в реальному часі (допомагає визначити інтереси користувачів та актуальність контенту).

Основними недоліками є [10]:

- потреба налагодженого зв'язку з мережею Інтернет;
- рівень безпеки (вразливість до SQL ін'єкцій, хакерських атак, порушення прав інтелектуальної власності).

Варто зазначити, що всі аналоги, представлені у підрозділі 1.2 реалізовані у формі веб-додатків або мобільних додатків, що пов'язано з простотою використання користувачами такого типу програмного забезпечення. Однак є і суттєва різниця між реалізацією веб-сервісу і мобільного додатку.

Мобільний додаток має ряд очевидних переваг [11]:

1. Специфіка архітектури дозволяє оптимізувати додаток під конкретну модель або серію телефонів.
2. Мобільний додаток може використовувати системні функції телефона та в автоматичному режимі мати доступ до додаткової інформації - поштової скриньки власника, журналу викликів, соціальних мереж, файлів тощо.
3. Розроблюваний інтерфейс може бути більш компактним та простішим за веб аналог.
4. Процес інтегрування набагато легший, універсальний інсталятор (для конкретної операційної системи) може встановити додаток.
5. Додаток може використовувати push-повідомлення, нагадуючи користувачу про новини, оновлення статусів тощо.
6. Мобільний додаток використовує ресурси телефону власника і не потребує серверної частини, за яку платить розробник аналогічного веб-сервісу.

Варто пам’ятати і про недоліки мобільних додатків, а саме:

1. Потреба підтримки особливостей моделі телефону, операційної системи, її версій.
2. Оптимізування для старих моделей телефонів.
3. Інтерфейс повинен коректно працювати та виглядати на різних моделях, а отже, і на різних розмірах екранів.
4. Постійні оновлення версій ОС часто спричиняють несправності в роботі мобільних додатків, а отже, потрібне постійне оновлення програмного забезпечення.
5. Доступ до сервісу обмежений однією людиною – власником телефону.

Оцінивши переваги та недоліки веб-сервісу та порівнявши з аналогічною характеристикою мобільних додатків, було обрано реалізацію

програмного забезпечення у формі веб-сервісу з використанням серверної (back end) та клієнтської частини (front end) задля можливості подальшого масштабування.

2.2. Вибір мови програмування для back end частини ПЗ

Однією з основних частин будь-якої програми є реалізація основної функціональності (спілкування з клієнтом), логіки, опрацювання і зберігання даних та архітектури. Частину проєкту, яка відповідає за вищеописані компоненти, називають back end [12]. Саме з проєктування та вибору мови програмування для back end кожен програміст починає розробку веб-сервісу, адже від цього залежить як метод інтегрування в інфраструктуру мережі інтернет, так і можливості та спосіб реалізації особливостей програми.

Наявність величезної кількості різноманітних мов для програмування back end зумовлена тим, що кожен проєкт націлений, зазвичай, на конкретний напрямок розробки – робота з великими обсягами даних, орієнтація на кінцевого користувача, безпека, швидкість розробки (орієнтування на програміста), кросплатформеність тощо. Одними з найпопулярніших та найбільш використовуваних є такі мови як JavaScript, включаючи різноманітні його фреймворки та бібліотеки, PHP (Laravel, який не втрачає популярність вже декілька років завдяки постійним оновленням програмного забезпечення), Python (особливо його фреймворк Django), Ruby в поєднанні з фреймворком Ruby on Rails, NodeJS та маловідомі – Elixir (і його легковісний і гарно оптимізований фреймворк Phoenix) та фреймворк Pyramid, завдяки відчутним відмінностям від стандартного вигляду та стилю програмування Python – виділений як окремий пункт. Всі вищеописані очолюють список найпопулярніших мов програмування back end у веб-сервісах як для початківців, так і для досвідчених програмістів, адже мають хорошу офіційну документацію та велику кількість різноманітних інструкцій та

настанов для реалізації будь-якої доступної функції. Популярність мови призводить до зростання кількості веб-застосунків, написаних на ній, а отже, і кількості коду, що зберігається у відкритому доступі на серверах github, gitlab та подібних. Це не означає, що всі мови програмування веб-сервісів однаково точно спроєктовані та використовуються. Всі наведені вище характеристики властиві для кожної з мов програмування в різній мірі, і тому є декілька конкретних прикладів. Ruby on Rails має детальну офіційну документацію та цілий цикл навчального матеріалу, PHP представлений великою спільнотою розробників (однією з найбільших на сервісах типу StackOverflow [13, 14]), які часто допомагають один одному. Java має декілька літературних видань, найвідомішим з яких є "Філософії Джава" Еккеля Брюса, де описані як базиси мови та технології для навчання, так і конкретні приклади та функції.

Головну роль при виборі мови програмування займає спеціалізація майбутнього веб-сервісу. Якщо для нього, чи його частини, головні швидкодія та оптимізація, варіантом є використання NodeJS [15] (висока швидкість та рівень продуктивності забезпечений потужним Chrome V8 Engine) чи Elixir (завдяки розмірам та легковагості технології Phoenix). Якщо питання швидкодії не є критичним, а сервіс не повинен підтримувати тисячі транзакцій в секунду, вибором може стати RoR чи Python. Варто зазначити, що Ruby з фреймворком RoR втрачає в швидкодії через структуру та модульність мови, а отже, при використанні чи відмові від певних бібліотек може показувати не гірший час за Elixir та NodeJS, в той час як Python має проблеми з використанням оперативної пам'яті, та при великих об'ємах її використання суттєво сповільнюється.

Другим пунктом при виборі мови написання веб-сервісу є так званий "поріг входу". Більшість веб-сервісів написані та підтримуються невеликими командами. Elixir та Ruby on Rails ідеально підходять для невеликих проєктів, адже вони "орієнтовані на програміста", що означає легке та зрозуміле написання коду на мові програмування, та не потребує

довгих архітектурних та інженерних досліджень. За рахунок наявності бібліотек [16] (гемів) повноцінний Ruby on Rails проєкт може бути написаний за декілька днів. Бібліотека взаємодії з клієнтом, яка використовує REST API (grape gem), бібліотеки для аутентифікації, зберігання захешовних паролів (doorkeeper, devise) – це невелика частина переліку найвідоміших бібліотек. Elixir виділяється простотою синтаксису, що в поєднанні з вседозволеністю коду, характерною для PHP, дає повну свободу програмісту в реалізації його ідей. Наявність такої еластичності зумовлює ризик появи проблем при майбутній підтримці та розробці, адже база для майбутньої функціональності могла не бути передбачена на початку (характерно для таких мов як Elixir, PHP).

Третім пунктом, який є актуальний для тих веб-сервісів, що в перспективі будуть мати багато користувачів, є масштабованість. Ця властивість проєкту характеризує можливість збільшення швидкості та пропускної здатності за рахунок збільшення використовуваних ресурсів. Найкращими в даному аспекті є Python [17], NodeJS та Ruby on Rails. Python, за рахунок архітектури мови (підтримки coroutines) має можливість асинхронно працювати та підтримувати декілька потоків, що не блокують один одного. NodeJS гарантує масштабованість у невеликих проєктах, за рахунок однопотокової асинхронної архітектури, що передбачає виконання операцій вводу та виводу за межами 23 потоку, а отже, без можливості його блокування. Однак, це характерно лише для невеликих проєктів. Ruby on Rails підтримує стандартні конфігурації кількості “робочих” нод та потоків: з’єднання з базою даних тощо. Зважаючи на це, а також на можливість збільшувати кількість реплікацій проєкту на RoR – додаток забезпечує майже будь-який потрібний рівень оптимізації за рахунок збільшення використовуваних ресурсів, а отже, буде обмежений тільки фізичними характеристиками системи (кількість RAM, процесорів тощо).

Наступним пунктом при виборі мови програмування є можливість та особливості опрацювання помилок. Найпростішими та

найуніверсальнішими в обробці помилок є JavaScript та Ruby on Rails. Вони мають повну підтримку приймання та обробки помилок усіх типів, і роблять це прозоро та легко. Налаштування дозволяють реагувати на конкретно визначений тип помилки, або ж ігнорувати цілий модуль таких. Варто зазначати, що за рахунок специфіки архітектури мови та наявних інструментів відлагодження – RoR, зазвичай, надає більше можливостей для пошуку та виправлення помилки.

Після проведення відповідності порівняльного аналізу до мети та функціональності майбутнього веб-сервісу мовою програмування для back end частини було обрано Ruby On Rails [18].

2.3. Аналіз технології розроблення front end частини ПЗ

Front end – це інтерфейс, тобто частина веб-сервісу, з якою безпосередньо взаємодіє кінцевий користувач. Раніше веб-розробка front end виконувалась людьми, які працювали з простими графічними редакторами та писали код на HTML та CSS [19]. З появою мови JavaScript можливості написання front end частини значно збільшились. На сьогоднішній день існує безліч різноманітних бібліотек та фреймворків для JS, що пропонують як стандартні функції, так і щось нове (оптимізацію, нову функціональність, структурованість тощо). Спільнота JS програмістів активно приймає участь в розробці нових бібліотек, пакетів, фреймворків (зараз npm, package manager JS, нараховує більше 600 000 пакетів, найбільше серед усіх пакетних менеджерів). Особливо виділяються: бібліотеки – React, D3 та фреймворки – Angular, Vue JS, Ember JS, адже вони визнані найрозповсюдженішими та найпопулярнішими.

React – це бібліотека JS, яку створили розробники Facebook і Instagram. Згідно опитувань серед розробників [20, 21], React – найпопулярніший вибір для написання інтерфейсу користувача (репозиторій [22] має найбільшу кількість зірок в Github серед усіх JS проєктів). Основними особливостями React є використання

декларативного підходу, який дозволяє контролювати стан проєкту, задаючи конкретні інструкції для представлення. Важливим аспектом є модульність архітектури. Згідно неї, кожен модуль має бути представлений власним компонентом, і чим менші ці компоненти, тим логічніше структурований сервіс, адже кожна окрема сторінка буде скомпонована з уже існуючих частин, кожна з яких має власний стан.

Віртуальна модель DOM вирішує проблему прямої маніпуляції, а однонаправленість потоку даних в сукупності з додатковим синтаксисом JSX і командною строкою дають можливість швидко та легко спроєктувати та написати React проєкт.

AngularJS – типовий представник фреймворків JS. Якщо React – це просто набір інструментів для JS, то особливості Angular змінюють його базові догми. Angular входить в топ-3 найвикористовуваніших інструментів розробки інтерфейсу користувача. Основна проблема фреймворку в складному процесі оновлення. Оновлення з 1 до 2 версії потребує дуже багато часу та технічних ресурсів, тому багато проєктів досі діють на базі першої версії. Серед особливостей можна виділити TypeScript – як мову над Angular по замовчуванню, а отже, і строгу типізацію, компонентну архітектуру (зазначимо, яка дозволяє більше свободи, ніж в React), оптимізацію та велику кількість інструментів та варіантів початкових каркасів.

VueJS – фреймворк, створений компанією Evan You, який набув популярності в основному через те, що більшість компаній та сервісів почали перехід від більш старих фреймворків та бібліотек як jquery, knockout, backbone. На сьогоднішній день кількість проєктів, які використовують фреймворк в якості основної мови програмування більше декількох десятків тисяч. VueJS особливий тим, що надає певну свободу розробнику в архітектурі програми. Vue не передбачає суворих правил написання, його легко вивчити та інтегрувати. Команда розробників фреймворку витратила багато часу та уваги для представлення веб-

розширень, які допомагають розробникам. З допомогою таких додатків програміст може перевірити поточні стани елементів та змінити деякі з них прямо в браузері. Важливим аргументом на користь Vue є його розмір. Фреймворк значно менший, а отже, і краще оптимізований, ніж його аналоги. Він не потребує додаткових бібліотек та може бути використаний як для написання окремих компонентів інтерфейсу, так і для повноцінних односторінкових веб-сервісів. Серед основних переваг VueJS можна виділити чіткість і простоту, наявність чітко поданої офіційної документації, просту інтеграцію та перевикористання коду завдяки логічній структурі. При цьому VueJS має і ряд недоліків, найважливішим з яких є невелика, в порівнянні з більш відомими react та angular, спільнота розробників.

Вищеописані інструменти лише частина з тих, що можуть займати місце мови реалізації інтерфейсу користувача. Однак, для того, аби проєкт можна було обслуговувати протягом декількох наступних років, потрібно бути впевненим в технології та її своєчасних оновленнях. Саме тому, остаточним вибором для front end розробки став React фреймворк.

2.4. Спосіб використання блокчейн технології для back end частини

За визначенням, блокчейн – це, перш за все, база даних, що зберігає впорядковані блоки (ланцюг записів), кількість яких невпинно росте з часом використання. Кожен наступний елемент містить хеш попереднього, дані транзакцій у вигляді хеш-дерева та позначку часу.

Незважаючи на те, що блокчейн тільки починають інтегрувати в існуючі системи, він вже отримав велику популярність. Обумовлено це, перш за все, можливостями, що відкриваються, а саме спрощення цілого ряду господарських, фінансових, а також бухгалтерських операцій [23]. В той же час, технологія необмежена конкретною сферою чи галуззю використання і може реформувати частини банківської, виборчої, судової, нотаріальної та реєстраційної системи тощо. Сервіси, що використовують

блокчейн могли б оптимізувати управління державою, адже грошова прозорість при укладанні смарт-контрактів (технології, що використовує блокчейн) – це основа відкритості та публічності, що пропагується усіма розвинутими країнами в останні роки. Потенціал блокчейн-сервісів практично невичерпний.

Смарт-контракт – це комп'ютерний алгоритм, основна функція якого, це укладання, зберігання та дотримання виконання умов комерційного відповідника контракту за допомогою технології блокчейн. Як вже було зазначено вище, "розумний" контракт і технологія блокчейн – пов'язані. Щоб забезпечити виконання контракту, перш за все, він має бути десь збережений. Саме для цього і використовується основна функція децентралізованої технології – блокчейн. "Розумний" контракт, що являє собою алгоритм дій, збережений всередині ланцюжку блоків блокчейн. Це дозволяє системі бути однозначною у трактуванні умов договору та дотримуватись їх виконання.

В основі роботи смарт-контракту є можливість збереження "стійкого цифрового ідентифікатору та особи" людини, що розблоковує можливість інтегрування технології у різні види та сфери людської діяльності [24]:

1. Логістика (повний контроль над процесом доставки та оплати за послуги).
2. Економіка (сплачення та отримання грошових переказів в автоматичному режимі в залежності від умов).
3. Захист інтелектуальних цінностей та прав власності.

Доцільно навести конкретні приклади можливого використання смарт-контрактів в сферах, близьких до людської діяльності.

Виборчий механізм – можна помістити результат голосування кожного виборця в блокчейн та копіювати між вузлами мережі. Варто зазначити, що при цьому всі дані анонімні та захешовані, а також незмінні. Такий спосіб повністю гарантує відсутність маніпуляцій над голосами виборців з моменту внесення даних в систему блокчейн.

Логістика – поставки, як правило, включають в себе складний процес, який схематично можна уявити у вигляді ланцюжка із багатьма ланками, де кожна ланка має отримати підтвердження від попередньої, виконати свою частину роботи та передати інформацію наступній. Цей процес знижує рівень оптимізації і збільшує витрати часу. Якщо використовувати смарт-контракт для зберігання і передачі інформації, це може зекономити час, а також дозволить відслідковувати прогрес в реальному часі, як для служби поставки, так і для кінцевого споживача.

Для створення смарт-контракту потрібно [25]:

1. Предмет договору – програма повинна мати доступ до товарів та послуг, з приводу яких заключається контракт, а також повинна мати можливість автоматично дати або закрити доступ до них сторонам контракту.
2. Цифрові підписи – коли учасники ініціюють договір, вони мають підписати його своїми приватними ключами.
3. Умови договору – умови смарт-контракта в формі операцій для виконання. Перелік і порядок операцій повинен в точності відповідати логічній формулі процесу контракту.
4. Децентралізована платформа – саме в блокчейні цієї платформи повинен записатись контракт і копії якого будуть зберігатись рівномірно на її вузлах.

Основні переваги смарт-контрактів [26]:

1. Стовідсоткова гарантія даних, що забезпечується за рахунок збереження у децентралізованому сховищі даних.
2. Відсутність посередників і безоплатність (зниження витрат за рахунок зниження кількості учасників процесу).
3. Точність (дані змінюються лише згідно умов та лише автоматично).
4. Непідкупність (після підписання умов та приведення в дію смарт-контракту – ніхто не взмозі втрутитись у його виконання,

якщо це не передбачено завчасно. Гарантія програми – її цілісність, що, на відміну від посередників, є непідкупною величиною).

5. Публічність (дані доступні публічно або за ключем, а отже, всі сторони, що приймають участь в угоді можуть перевірити статус угоди. Проте сторони зберігають повну анонімність та конфіденційність інформації).
6. Монолітність (блокчейн – децентралізована технологія, яка не має єдиного серверу даних. Копії зберігаються на всіх комп'ютерах користувачів з усього світу, а отже, єдиним, теоретично можливим способом зміни чи знищення даних – був би контроль над усіма електронними пристроями світу та мережею інтернет. На практиці це, звісно, неможливо, а отже, і дані – в цілковитій безпеці.
7. Оптимізованість – так, як всі процеси протікають автоматично і не залежать від людського фактору – виконання смарт-контракту займатиме не більше кількох хвилин, в той час як аналог, що використовується зараз займає період в десятки, сотні разів довший.

Серед недоліків варто визначити такі пункти:

1. Практична складність реалізації (сучасна система інертна, для того аби її реформувати потрібно задіяти багато спеціалістів, поступово вводити в дію та ознайомлювати з нею всіх членів процесу, а це, зазвичай, потребує багато часу, грошей та зусиль).
2. Архітектура має бути досконала (кожна умова контракту має бути продумана та записана перед його укладанням. Якщо буде допущена помилка в програмно коді, в логіці протікання договору чи в особистих даних – змінити їх вже буде неможливо).
3. Фізична відсутність регулятора.

Зазвичай, після досягнення фізичної згоди, розумний контракт треба підписати в системі (рис. 2.1). Для цього використовуються так звані механізми підписів, які можуть підтвердити транзакцію в мережах криптографічних валют. Для того, аби контракт корректно спрацював і реалізувався в блокчейні потрібно, аби він мав правильну математичну та формальну логіку, задану програмним кодом.

Такий договір вважають легітимним, якщо досягається взаємно згода сторін. Існує єдина форма документа, згідно якої договори такого типу і укладають. Для скріплення контракту ставляться підписи сторін або проходить обмін листами, що є спрощеною формою. Процес протікає автоматично, система зберігає та виконує умови “розумного” контракту, а також відстежує дії сторін. Зазвичай алгоритм передбачає блокування контракту, якщо одна або декілька умов не виконані.

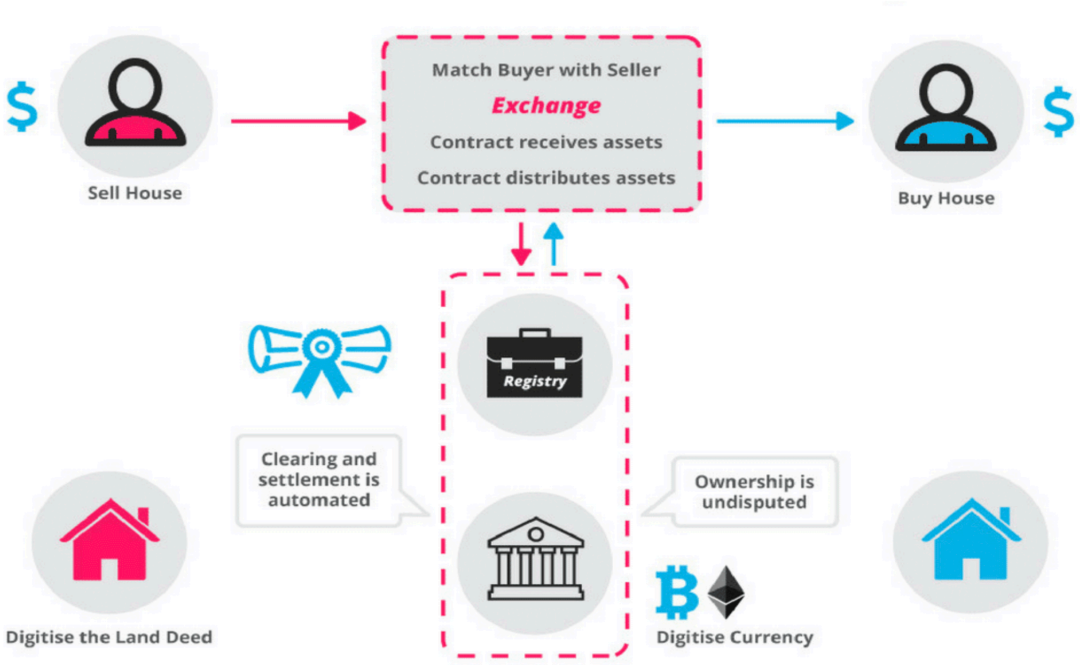


Рис. 2.1. Принцип роботи смарт-контракту [27]

Підсумовуючи все вище описане – зручність користування, високий рівень безпеки, зашифрованість та конфіденційність даних, математична точність та надійність є основними перевагами розумних контрактів.

3. СТРУКТУРНО-АЛГОРИТМІЧНА ОРГАНІЗАЦІЯ

3.1. Загальна структура системи

Система, як веб-сервіс, складається з 2 частин (front end та back end). Front end також називають клієнтською частиною, тобто такою, з якою безпосередньо взаємодіє кінцевий користувач, back end – серверна частина, яка відповідає за виконання основних функцій програми. Кожна з вище описаних частин веб-сервісу виконує ряд завдань згідно архітектурного та функціонального плану розробки та відповідних вимог.

Функціональність back end частини можна виділити в декілька пунктів, а саме:

1. Спілкування з front end частиною HTTP запитами та обробка отриманих даних.
2. Реалізація стилю архітектури REST для розподілених систем на основі фреймворку ROR.
3. Реалізація CRUD (створення, читання, оновлення, видалення) взаємодії з базою даних.
4. Збір та зберігання інформації про користувачів, об’єкти угод, поточні угоди користувачів та їх відповідний стан.
5. Реалізація смарт-контракту та запис даних у мережу блокчейн.
6. Моніторинг мережі блокчейн для підтримання актуальності даних про угоди.

Функціональність front end частини можна описати в декількох тезах, а саме:

1. Реалізація спілкування на рівні система-користувач завдяки розробленому інтерфейсу користувача (UI).
2. Відправка даних для їх подальшої обробки на серверну частину.
3. Реалізація візуального відповідника поточного статусу виконання смарт-контракту, його укладення.

Веб-сервер – це модульна програма, а отже, його структурна схема (див. Додаток 1), складається з декількох модулів, кожен з яких виконує свій ряд функцій та залежить від інших частин програмного забезпечення. Згідно наявних елементів програми можна виділити такі типи модулів: база даних, функціональні модулі, модуль користувачів системи.

В даному програмному забезпеченні можна виділити такі конкретні модулі програми:

1. Модуль базової реєстрації користувача. Збір основних даних про користувача (електронна пошта, пароль) та їх збереження для подального використання. Представлений контролером REST API “/identity/users”. Включає в себе також набір функцій: підтвердження електронної пошти для закріплення за користувачем, відновлення та зміна паролю.
2. Модуль сесії користувача. Відповідає за відкриття та збереження даних про поточну сесію користувача та містить інформацію про нього. Представлений контролером REST API “/identity/sessions”.
3. Модуль інформації та KYC. Ця частина проєкту може бути умовно поділена на такі підмодулі: інформація про користувача та його роль, модуль телефонів, профілів, документів. Саме цей модуль розділяє користувачів на потенційних орендарів та орендодавців. Модуль телефонів відповідає за збереження інформації про телефонний номер, його підтвердження через канал смс/дзвінок, модуль профілю відповідає за збір та збереження інформації про користувача (його ім’я, прізвище, дата народження тощо), а модуль документів відповідає за збір даних (у форматі зображення) про офіційний документ, що посвідчує особу (паспорт, водійські права). Модуль представлено REST API “resource” та підмодулями “phones”, “documents”, “users”, “profiles”.

4. Модуль історії операцій. Ця частина програми відповідає за зчитування та представлення даних про минулі та поточні угоди користувачів, їх статистику, дати та всю супутну інформацію.
5. Модуль статистики та адміністрації. Цей модуль реалізує адміністративну функціональність, де адміністратор може побачити список всіх користувачів, відфільтрувати його за типом користувачів, побачити детальну інформацію щодо їх профілів, документів тощо.
6. Модуль операцій орендодавця. Відповідає за збір інформації від користувача про об’єкт оренди та зберігає її для майбутнього опрацювання та представлення в списку доступних об’єктів для оренди. Тісно взаємодіє з базою даних.
7. Модуль укладення угод. Відповідає за збір інформації та зв’язує користувача (орендара), об’єкти оренди, умови угоди. Передає отриману інформацію для опрацювання у модуль смарт-контрактів.
8. Модуль смарт-контрактів. Реалізує смарт-контракт та спілкування з мережею блокчейн. Використовує інформацію отриману від користувачів про умови угоди та об’єкт оренди, а також інформацію про самих користувачів.

Кожен з вищеописаних модулів використовує чи передає інформацію іншим. За рахунок взаємодії реалізується основна функціональність програми.

3.2. Структура даних систем

Для того, щоб задовольнити основні вимоги проєкту, веб-сервіс тісно взаємодіє з базою даних, схема якої була спроєктована згідно плану розробки та містить такі таблиці – Users, Properties, Profiles, Phones, Permissions, Orders, Levels, Labels, Documents, Accommodations, Attachments (див. Додаток 1). Таблиці нормалізовані за змістовою ознакою,

що означає, що вони містять окремі набори даних, що використовуються у записах.

Таблиці Documents, Profiles, Labels, Orders та Properties пов’язані з таблицею Users зовнішнім ключем user_id та отримали індекс на це поле, для пришвидшення виконання запитів пошуку відповідного користувача / його суб’єкта. Згідно основних базисів мови програмування Ruby та фреймворку Ruby on Rails таблиця Users отримала такі відношення: “has_many: labels, documents, orders, properties, phones”, “has_one: profile”. Таблиця Users зберігає основну інформацію про користувача та його особисті дані:

- uid – унікальний ідентифікатор користувача;
- email – адреса електронної пошти користувача;
- password digest – захешований за допомогою технології bcrypt пароль користувача;
- role – роль користувача (початкове значення – member, звичайний користувач, доступні – landlord, super admin, member);
- level – рівень користувача згідно KYC (know your customer);
- otp – показник використання двофакторної аутентифікації;
- state – статус акаунту користувача;
- referral id – посилання на ідентифікатор реферала;
- eth address – ETH адреса користувача;
- created at – час створення запису;
- updated at – час останньої зміни даних запису.

Таблиця Levels зберігає інформацію про кількість та вимоги до рівня користувача:

- key – назва необхідної для отримання рівня мітки;
- value – значення необхідної для отримання рівня мітки;
- description – опис рівня, способу його отримання чи загальної інформації про нього.

Таблиця Labels зберігає інформацію про мітки користувача, відповідно, також містить посилання на запис таблиці Users:

- key – назва мітки;
- value – значення мітки;
- scope – показник статусу мітки, можливі значення – private, public, тобто системна, або публічна мітка;
- description – коментарі до мітки.

Таблиця Phones містить інформацію про дані телефону користувача. Містить посилання на таблицю Users, адже між ними реалізовано відношення “belongs_to”:

- country – ідентифікатор країни (перші 2-3 цифри номеру);
- number – мобільний номер користувача;
- code – код, для підтвердження власності номеру;
- validated at – час підтвердження телефону.

Таблиця Profiles зберігає інформацію про особисті дані користувача, що приймають участь в процесі KYC. Містить посилання на таблицю Users, адже між ними реалізовано відношення “belongs_to”:

- first name – ім’я користувача;
- last name – прізвище користувача;
- dob – дата народження;
- address – адреса проживання;
- post code – поштовий індекс;
- city – місто проживання;
- country – країна проживання.

Таблиця Documents – містить інформацію про представлений користувачем документ, що підтверджує особу. Містить посилання на таблицю Users, адже між ними реалізовано відношення “belongs_to”:

- upload – завантажений користувачем файл (фото) документу;
- doc type – тип документу (водійські права, паспорт);
- doc number – номер документу;

- doc expire – кінцева дата дії документу, згідно закону країни, що видала його.

Таблиця Permissions – ця таблиця конфігурує доступ до API згідно ролі користувача.

- action – показник дії, що встановлюється цим правилом (ACCEPT / REJECT);
- role – роль користувача, на яку впливає дане правило;
- verb – тип запиту, на який впливає дане правило (GET, POST, PUT, ALL);
- path – шлях до API, або його початок (ідентифікатор модулю), на який впливатиме дане правило;
- topic – загальний коментар до правила, призначено для адміністраторів, аби виокремити правила в окремі групи.

Таблиця Properties – містить основну інформацію про об’єкт чи групу об’єктів оренди. Містить посилання на таблицю Users, адже між ними реалізовано відношення “belongs_to”, а також має “has_many” зв’язок з таблицею Accommodations.

- kind – тип місця, що здається в оренду (приватний будинок, готель, мотель, вілла);
- name – назва нерухомості, що здається в оренду;
- address – адреса місцезнаходження закладу;
- picture – зображення нерухомості;
- description – основна інформація про нерухомість, що здається в оренду;
- eth address – адрес гаманця орендодавця, на який будуть переведені гроші за оренду.

Таблиця Accommodations – зберігає інформацію про безпосередній об’єкт, що здається в оренду. Містить посилання на таблицю Properties, адже між ними реалізовано зв’язок “belongs_to”:

- name – публічний ідентифікатор, назва кімнати, будинку або приміщення;
- kind – тип об’єкту оренди (кімната, будинок, поверх);
- capacity – рекомендована вмістимість (кількість людей);
- price – вартість оренди за один день;
- attachment – додаткова інформація про наявність чи відсутність побутових речей тощо;
- state – поточний статус об’єкту (зайнято, здається, недоступно).

Таблиця Orders містить інформацію про запити на оренду. Містить посилання на таблицю Users, адже між ними реалізовано відношення “belongs_to”:

- state – статус оренди (планується, в процесі, відхилена, завершена);
- eth address – адреса смарт-контракту;
- arrive at – дата початку оренди;
- departure at – дата кінця оренди.

Таблиця Attachments містить зображення про об’єкт, який здається в оренду. Містить посилання на таблицю Accommodations, адже між ними реалізовано відношення “belongs to”:

- upload – зображення об’єкту оренди (кімната, будинок, поверх).

3.3. Архітектура системи

Клієнтська частина реалізована за допомогою моделі single page application [28], основною особливістю якої є оновлення даних без перезавантаження сторінки. Технологія використовує AJAX-запити, які відбуваються під час, або внаслідок дій користувача та дозволяють отримувати інформацію від серверу динамічно, а отже, і оновлювати її. Така

модель вже набула популярності та стала стандартом веб-сфери інформаційних технологій, тому саме вона була обрана для користувацької частини проєкту. Щодо архітектури в цілому, було вирішено дотримуватись концепції мікросервісів [29], популярного типу інфраструктури проєктів. Мікросервісна архітектура в цьому випадку дозволяє цілій системі легко масштабуватись та оперуватись.

Остаточна архітектура веб-сервісу представлена схематично (рис. 3.1)

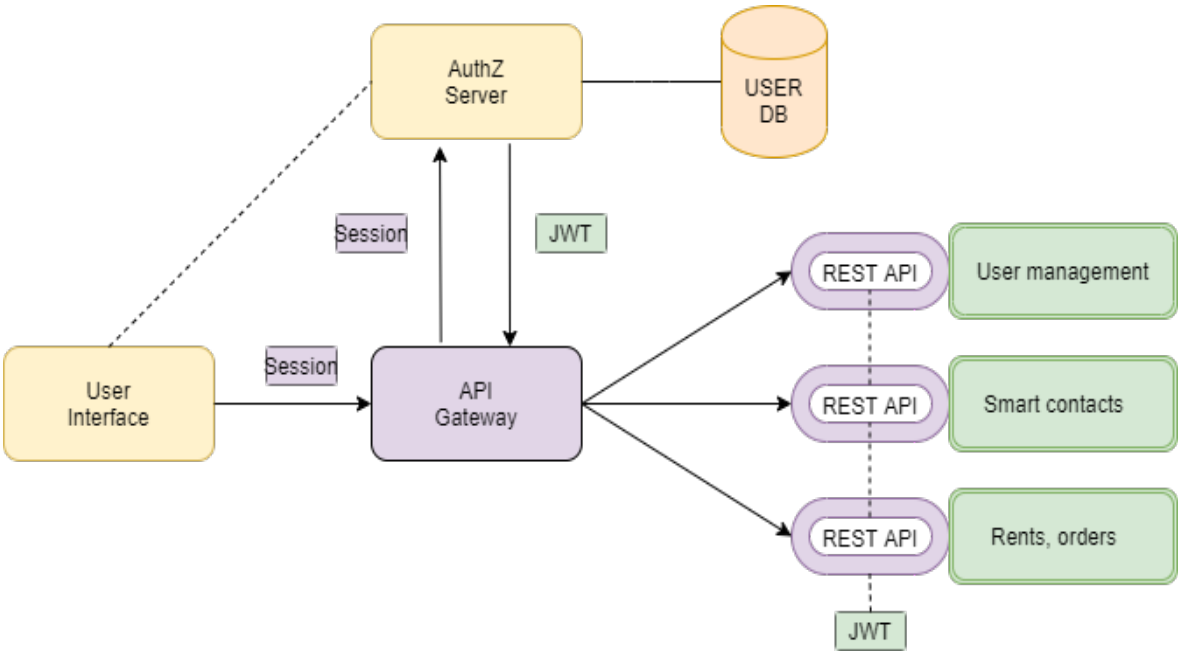


Рис. 3.1. Архітектура системи

Розглядаючи архітектуру більш детально, варто звернути увагу на процес авторизації та захисту API-серверів. Як вже було зазначено вище, клієнтська частина (UI) використовує HTTP-запити для оновлення даних, але запити перед потраплянням на сервер, спочатку проходять мікросервіс, що має назву API Gateway [30].

Реалізовано API Gateway компонент на базі Envoy – L7 проксі сервісу. Gateway робить систему прозорою, адже представляє собою “шину” між клієнтською та серверною частиною, перенаправляє запити до відповідного серверу, дозволяючи серверу приймати та відправляти запити на “localhost”,

тобто спілкуватись із локальною мережею, та не маніпулювати даними про публічну частину мережі.

Gateway дозволяє комбінувати мікросервіси, реалізовані на різних мовах програмування або фреймворках, адже вони не взаємодіють один з одним напряму, а отже, і не потрібна підтримка сумісності. Envoy ідеально підходить під поставлене завдання, адже він розроблений на C++11 і оптимізований для того, аби використання Gateway не зумовило втрату швидкості виконання запитів для кінцевого користувача. Envoy не обмежується описаними вище особливостями. Envoy виступає, як мережевий проксі L3/L4 і дозволяє записувати фільтри для виконання завдань TCP, а отже, і вставляти їх на сервер. Також Envoy підтримує додатково архітектуру фільтрів HTTP L7, що дозволяє виконувати завдання буферизації, переадресації або маршрутизації. Під час роботи в режимі HTTP Envoy підтримує і HTTP 1.1, і HTTP 2, а отже, може працювати як прозорий проксі-сервер HTTP 1.1 до HTTP 2 в обох напрямках.

В даному проєкті Envoy також використовується для перевірки стану компонентів, так званого “Health Checking”. Envoy підтримує як активну фазу перевірки стану компонентів, так і пасивну, що працює та виконується увесь час. Оскільки Envoy є автономним проксі-сервером, то він приймає участь також у розподіленні навантаження та підтримує методи балансування: автоматичних повторних спроб, розриву ланцюга, обмеження кількості запитів та виявлення зовнішньої швидкості. Зважаючи на все зазначене вище – API Gateway, реалізований на базі Envoy, ідеально підходить в заплановану мікросервісну архітектуру.

3.4. Модуль авторизації

Важливим, та особливим аспектом архітектури є процес авторизації. Проєкт комбінує два типи авторизації – за допомогою файлів cookies та веб-токену JWT. Користувач під активної сесії зберігає лише файл cookies, який

надсилається з кожним запитом веб-переглядачем користувача. В той же час сервер очікує саме JWT з інформацією про користувача. А заміна cookies на JWT проходить в мікросервісі, що носить назву – Authorize [31], куди Gateway робить запит на кожен користувацький запит на приватні API.

Весь процес разом з технічними деталями можна описати так:

1. Користувач вводить свої облікові дані.
2. Сервер перевіряє правильність облікових даних та створює сесію, яка зберігається в Redis, та формує cookies, що відправляється користувача у вигляді заголовку “Set-Cookie”.
3. Файл cookie з сесією розміщується у веб-переглядачі користувача.
4. При запиті на приватні API (конфігурація та розподілення шляхів API на приватні та публічні розміщується як в API Gateway, так і в Authorize – подвійний захист) API Gateway перенаправляє запит на Authorize, де перевіряється наявність ідентифікатору сесії та її вміст, а також порівнюється з даними, що знаходяться в Redis.
5. Якщо дані правильні, формується JWT з останньою наявною в базі даних інформацією про користувача та повертається на Gateway у вигляді заголовку “Authorization Bearer”.
6. Gateway перенаправляє запит на кінцевий сервер, де перевіряється інформація наявна в Authorization Bearer та її власник (за приватним ключем).
7. Якщо інформація підтверджується, виконується кінцева функція, а відповідь повертається на API Gateway.
8. API Gateway фільтрує заголовки та дані згідно конфігурацій та повертає відповідь кінцевому користувачу.

Незважаючи на складний та поетапний процес із багатьма учасниками, система працює швидко, гарно оптимізована та добре захищена, а за рахунок постійного оновлення JWT – сервер має останні дані

про користувача, що дозволяє динамічно реагувати на зміни його стану чи даних.

За API Gateway та Authorize розташовується група сервісів, доступ до яких кінцевий користувач отримує завдяки HTTP-запитам на API сервер. В даному проєкті використовується REST API, тобто набір правил до API endpoint. Ці правила визначають, як виглядає API. Одне з правил передбачає, що користувач повинен мати можливість отримати фрагмент даних (ресурс), коли він посилається на певну URL-адресу. Описана вище група серверів також має доступ до єдиної реляційної бази даних, де зберігаються всі дані, що використовуються проєктом.

3.5. Алгоритм роботи смарт-контракту

Як було зазначено вище, смарт-контракт – це комп'ютерний алгоритм, що призначений для формування, контролю стану та збереження інформації про власність, використання тощо. Смарт-контракт обов'язково повинен мати об'єкти, а саме:

1. Сторони договору – в даному проєкті це орендар (звичайний користувач, що знайшов вільне житло через веб-сервіс та зробив замовлення) та орендодавець (користувач веб-сервісу, який пропонує своє житло в якості оренди та володіє безпосередньо тим, яке вибрав орендар).
2. Предмет договору – житло у власності орендодавця, яке він виставляє для оренди.
3. Умови – умовами смарт-контракту в даному випадку є ціна оренди, час запланованого перебування, наявність необхідного побутового приладдя та описаний стан житла та речей у ньому. Можливі також додаткові умови, в залежності від конкретного житла та сторін договору.

4. Децентралізована платформа – блокчейн мережа Ethereum. Такий вибір зумовлено мовою написання смарт-контракту – Solidity.

За наявності усіх об’єктів контракту він може бути приведений в дію. В даному проєкті, як вже було зазначено вище, мовою написання смарт-контракту виступає Solidity. Solidity – це статично типізований, контрактно-направлена мова програмування, що синтаксично схожа на JavaScript (ECMAScript), але має статичну типізацію змінних та динамічні типи значень, що повертаються. Програми, написані цією мовою транслюються в байт код EVM (віртуальна машина Ethereum) і результатом їх дії є записи у вигляді транзакцій у мережі блокчейн.

Отже, всі інструменти для того, аби розробити смарт-контракт вже вибрані та доступні в межах даного веб-сервісу. Основний принцип роботи веб-сервісу та смарт-контракту можна описати поетапно.

1. Користувач платформи, що шукає житло (орендар) обирає один із доступних варіантів.
2. Смарт-контракт створюється та записується (deploy) в мережу блокчейн.
3. Гроші з адреси (на платформі) орендаря надсилаються на адресу смарт-контракту в якості оплати (в ETH згідно вказаної ціни та курсу). Як пароль користувач повинен вказати номер документу, що посвідчує особу, з яким він має прибути на фактичне місце на початку часу оренди (номер буде захешовано згідно SHA - 256).
4. Якщо сума переведена на адресу гаманця достатня – це рахується фактом передплати та бронювання.
5. На початку бронювання, коли орендар зустрічається з власником житла для отримання ключів/картки доступу до житла – він пред’являє документ, номер якого вказав як пароль до гаманця.

6. Орендодавець підтверджує фізичний заїзд орендаря кошти, за допомогою з номеру документу орендаря (який буде захешовано під час валідації та порівняно з хешем вказаним при створенні). Цей момент є фактичним початком оренди.
7. Під час виселення згідно запланованої дати орендар третім підписом підтверджує факт завершення оренди. Отже, multisig працює за принципом – три підписи із трьох. Перший – підтвердження про оплату, другий – номер документу орендаря, третій – орендар підтверджує факт закінчення оренди.
8. Після дотримання правила 3 підписів, гроші будуть переведені на рахунок орендодавця.

4. АНАЛІЗ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Особливості реалізації

Back end частина веб-сервісу реалізована у вигляді API-серверу, а отже не має інтегрованої візуальної частини. З користувачем (front end частиною) сервер комунікує за допомогою HTTP запитів (POST – для створення об'єктів, PUT – для зміни, GET – для "читання" та інші). Така чітка класифікацію відповідає архітектурному стилю REST (Representational State Transfer), основою якого є представлення ресурсу через його стан, що є майже дослівним перекладом абревіатури. Дизайн спілкування клієнтської та серверної частини схематично можна побачити на рис. 4.1.

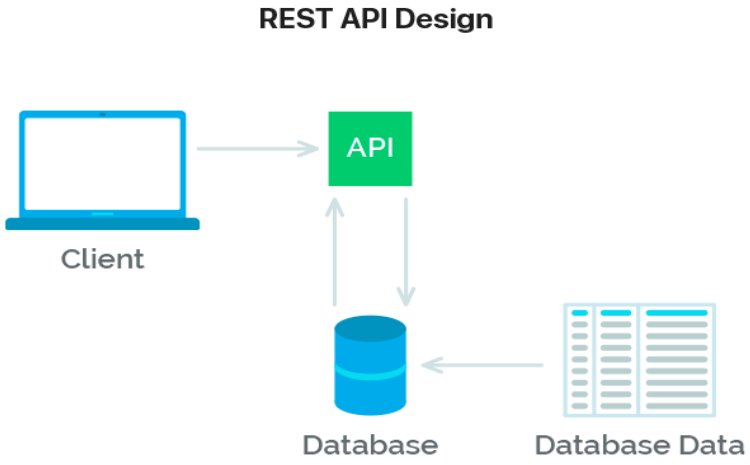


Рис. 4.1. REST спілкування між клієнтом та сервером

Незважаючи на те, що основною мовою програмування серверної частини сервісу є Ruby в поєднанні з фреймворком Ruby on Rails, для якого характерна архітектура MVC (Model-View-Controller), було обрано саме RESTful. Такий вибір обумовлено тим, що MVC підходить монолітним проєктам, в той час як REST дозволяє притримуватись поділу на мікросервіси. Такий стиль дозволяє побудувати частини веб-додатку, що легко масштабуються та розвиваються незалежно одна від одної та

спроєктувати такий набір кінцевих точок (API endpoints), який буде легко підтримувати та комфортно використовувати клієнтським пристроям, незалежно від їх мови програмування.

Ruby on Rails має вбудовану підтримку API контролерів (Rails API, Rails Metal), але в даному проєкті використана бібліотека Grape::API, яка є REST-подібним інтерфейсом для Ruby. Основною перевагою Grape є надання зручної DSL (domain specific language) для розробки API. В той час як можливості вбудованих помічників Rails дуже мінімізовані – Grape має модулі підтримки валідацій типу та формату даних, що отримуються та повертаються, версій компонентів, поділу на логічні модулі та автоматичного будування шляхів API endpoints, а також додаткові модулі для тестування та підтримки журналу виконання (логів проєкту).

Основна бізнес-логіка проєкту реалізована у моделях та сервісах у вигляді набору функцій з відповідними валідаціями та залежностями (рис. 4.2). Виклик цих функцій відбувається у Grape API контролерах, які і містять API endpoints як спосіб спілкування з сервером. Дані, що надходять у контролери та повертаються з них контролюються структурами даних для відповідних об’єктів (Grape entity), а отже, можливість шахрайської маніпуляції даними майже виключена.

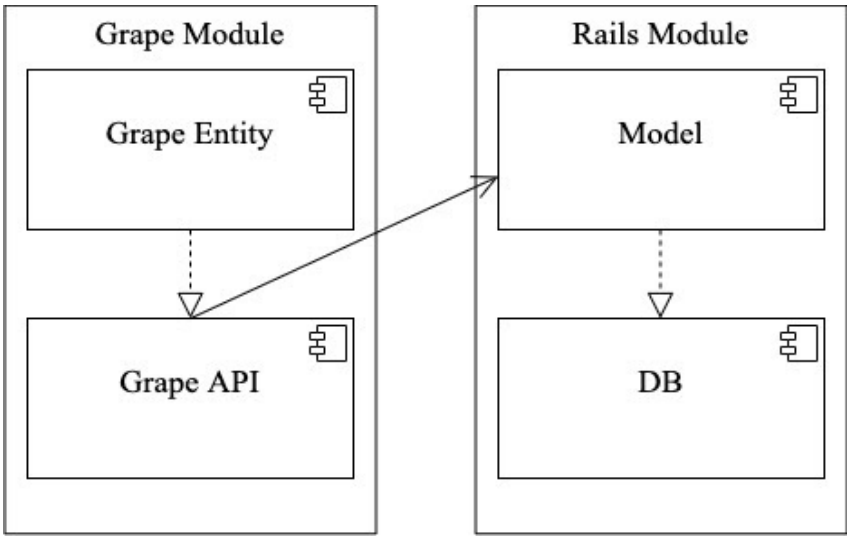


Рис. 4.2. Схема взаємодії бібліотеки Grape та Rails

4.2. Дизайн та вміст веб-сторінок

Клієнтську частину (front end) реалізовано мовою Javascript, конкретніше, її бібліотекою React. Також використовується HTML, CSS (PCSS). Наявний інтерфейс представлений двома незалежними модулями – адмінським (admin і landlord підмодулі) та для звичайних користувачів платформи.

Адмін частину імплементовано з використанням Material UI фреймворку, який має власний набір вже готових інструментів та компонентів, за допомогою яких можна полегшити проєктування власного дизайну або використати вже наявний. Так як основна мета адмінського модулю полягає саме в наданні функціональності, то використання вже запроєктованого дизайну адмін панелі, що відповідає основним стандартам UX видається логічним.

Компонент має такі сторінки:

Модуль “Адмін”:

1. Сторінка з загальною інформацією та метриками.
2. Сторінка з переліком користувачів платформи.
3. Сторінка з детальною інформацію про користувача.
4. Сторінка з документами KYC користувача.
5. Сторінка з переліком активних замовлень користувача.
6. Сторінка з історією замовлень користувача.
7. Сторінка з інформацією про активність користувача.
8. Сторінка з переліком документів KYC, що очікують підтвердження.

Модуль “Орендодавець”:

1. Сторінка з переліком об’єктів оренди (property).
2. Сторінка додавання нового об’єкту оренди (property).
3. Сторінка перегляду об’єкту оренди (property) та зміни його інформації.
4. Сторінка з переліком об’єктів розміщення (accommodation).

5. Сторінка додавання нового об’єкту розміщення (accommodation).
6. Сторінка перегляду об’єкту розміщення (accommodation) та зміни його інформації.
7. Сторінка з переліком замовлень, що відносяться до об’єктів оренди у власності (активні та неактивні).

Особливістю адміністративної панелі є те, що компонент має різний вигляд в залежності від ролі авторизованого користувача. Реалізовано це завдяки відповідним валідаціям на сервері та різній навігаційній панелі користувацького інтерфейсу. Візуальну різницю між навігаційними панелями адміністратора платформи і орендодавця зображено на рис. 4.3.

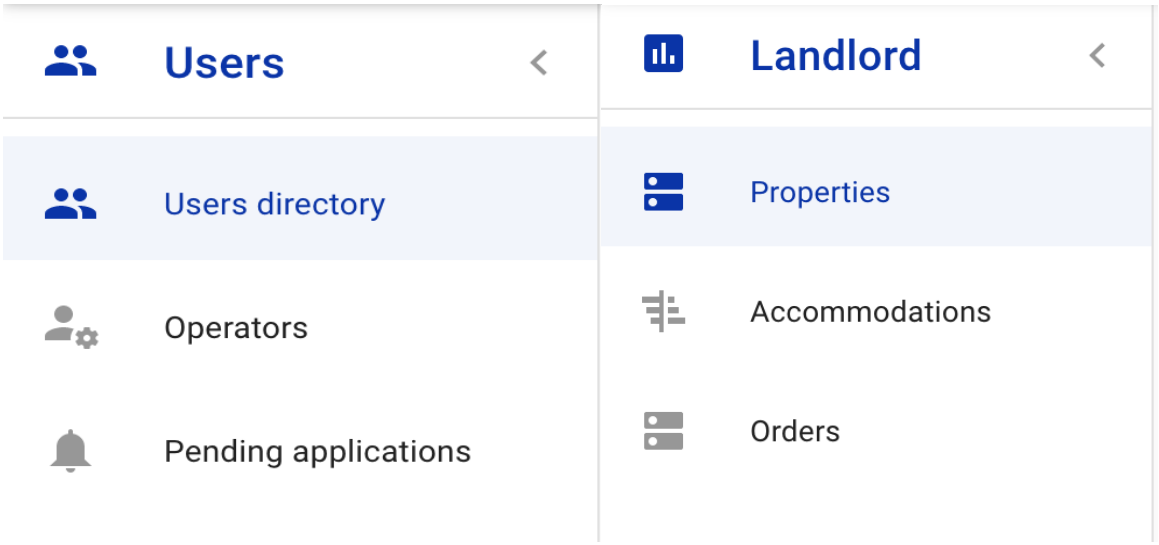


Рис. 4.3. Види навігаційної панелі

Важливим є постійний контроль за платформою, активністю, кількістю нових користувачів. Адміністратори платформи мають доступ до сторінки з метриками, на якій у вигляді діаграми зображена кількість активних користувачів (кількість авторизацій), кількість нових користувачів (кількість реєстрацій) та кількість невдалих спроб авторизації. Дані на графіку згруповані по дням. Дана сторінка зображена на рис. 4.4.

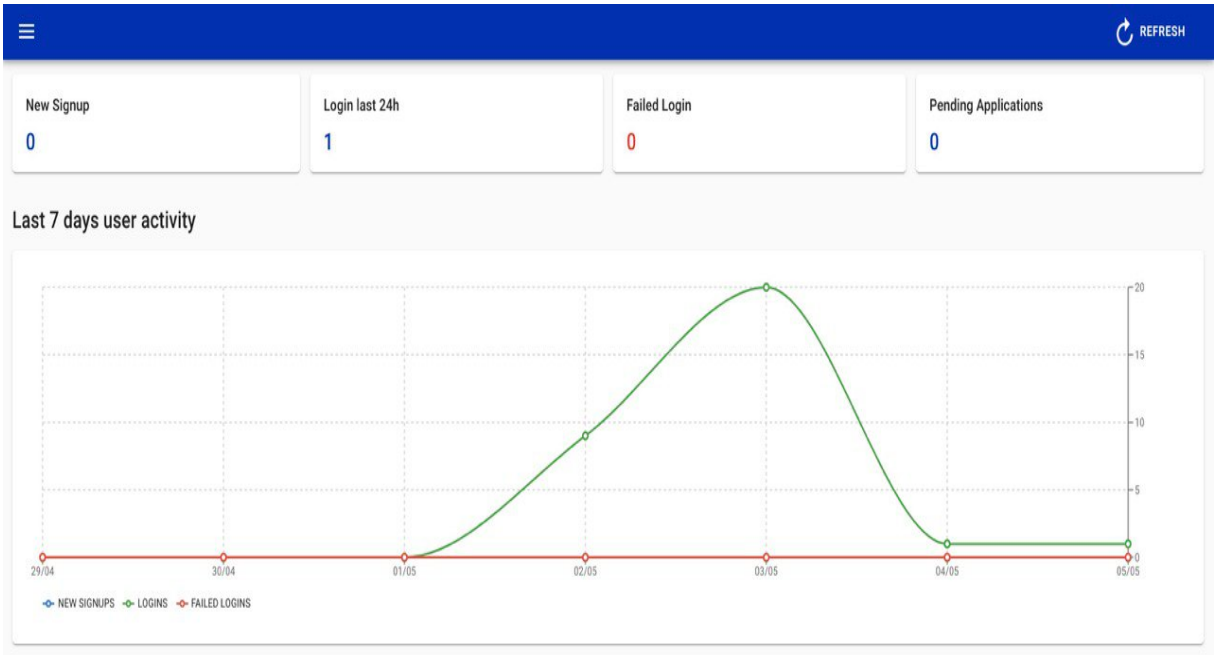


Рис. 4.4. Статистичні дані активності платформи

Одним з головних аспектів будь-якої платформи є контроль користувацьких акаунтів. Rentware забезпечує адміністратора всіма потрібними інструментами для цього. Адміністратор може переглянути список всіх користувачів, зареєстрованих на платформі, переглянути коротку інформацію (таку як електронна адреса, ім’я, роль, рівень, країна, статус та час створення акаунту), а також повну інформацію, включно з профілем користувача (рис. 4.5). На цій самій сторінці адміністратор може змінити роль користувача, статус його акаунту, рівень КУС. В разі потреби адміністратор може вручну створити позначку (як приватну, системну, так і публічну, користувацьку) з відповідною до потреби назвою та значенням, а також описом. За позначками адміністратор може виділяти групи користувачів за допомогою фільтрів тощо. Адміністратор також має змогу вимкнути двофакторну аутентифікацію відповідного акаунту.

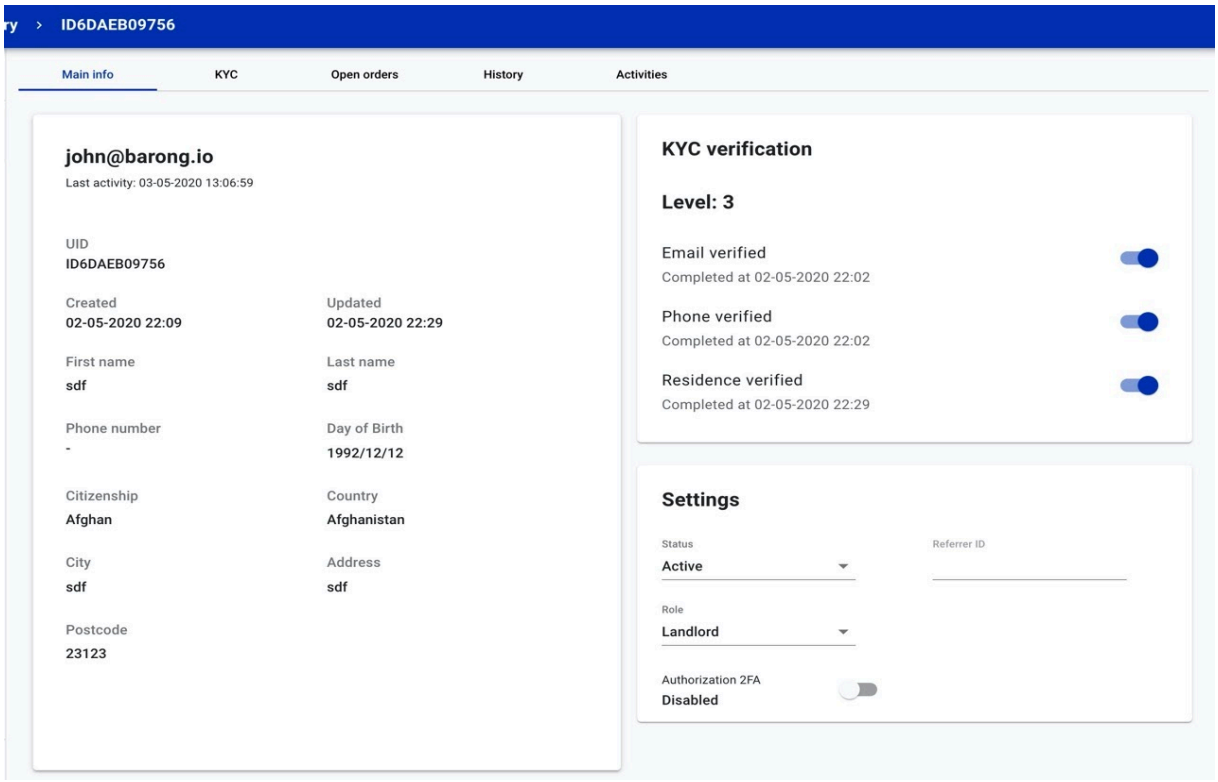


Рис. 4.5. Сторінка інформації про користувача

Орендодавець, в свою чергу, за допомогою адміністративної панелі може контролювати свої об’єкти, переглядати список об’єктів оренди у власності, переглядати та редагувати детальну інформацію про них, додавати та переглядати фотоматеріали, закріплені як за об’єктом оренди (property) в цілому, так і за об’єктами розміщення (accommodation). Дизайн сторінки списку об’єктів оренди зображено на рис. 4.6.

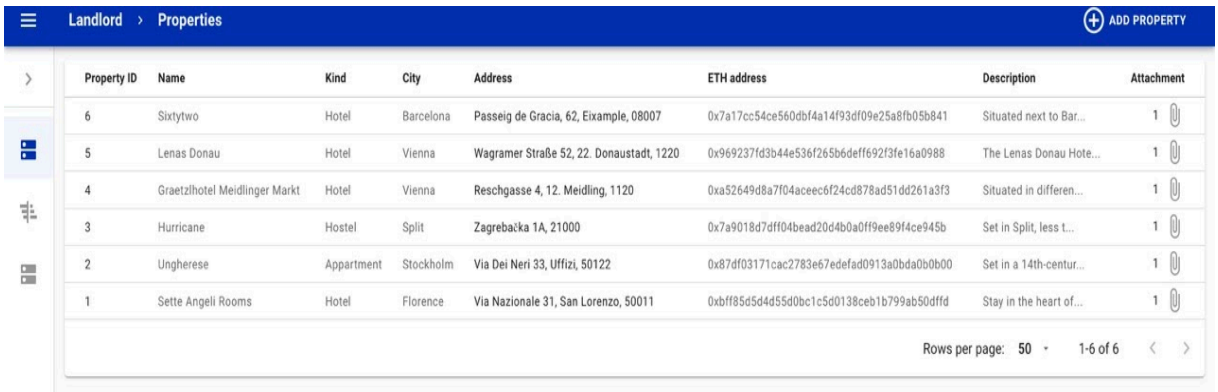


Рис. 4.6. Список об’єктів оренди

Важливим для орендодавця є можливість слідкувати за поточними та запланованими запитами на оренду. Таку функціональність реалізовано на окремій сторінці Orders, до якої орендодавець має доступ через навігаційну панель. Список реалізовано у вигляді таблиці, що містить основну інформацію по кожному запиту на оренду. Остання колонка таблиці зарезервована під “unlock” кнопку, яка з’являється лише для активних замовлень та при натисненні викликає модальне вікно з текстовим полем, в яке орендодавець повинен ввести правильний пароль (номер документу тощо), пред’явленого користувачем під час фізичного заїзду (рис. 4.7).

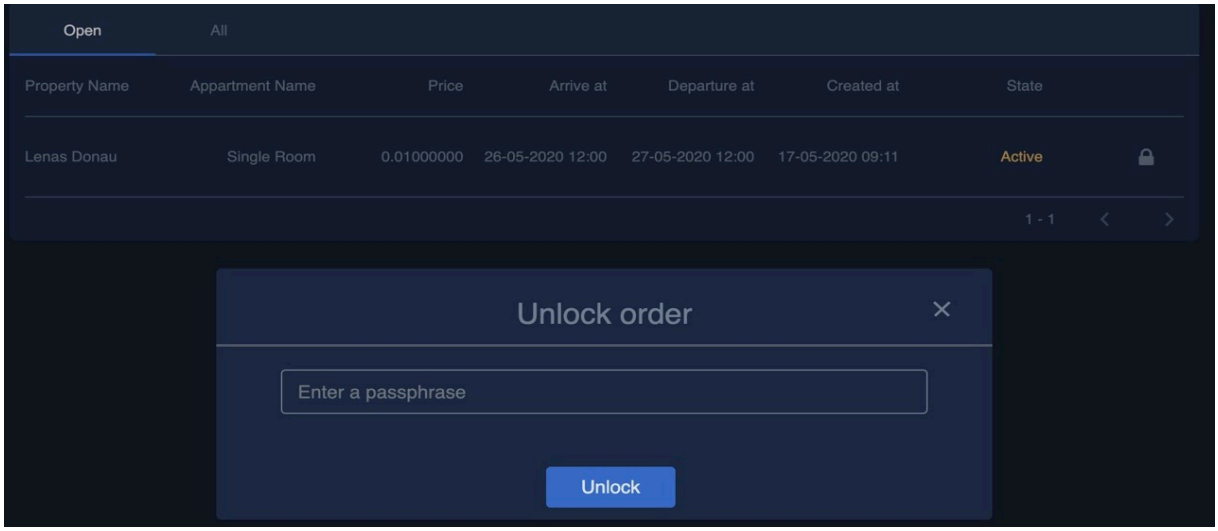


Рис. 4.7. Розблокування замовлення

Другим компонентом, що представляє front end частину, є модуль користувача – основа взаємодії з сервером. Основою реалізації став React у поєднанні з Typescript, також були використані деякі bootstrap модулі для забезпечення адаптивності інтерфейсу під різні розміри та роздільні здатності екранів та девайсів.

Користувацька частина включає в себе такі сторінки:

1. Сторінка реєстрації.
2. Сторінка авторизації.
3. Сторінка пошуку об’єктів оренди.
4. Сторінка детальної інформації про об’єкт оренди.

5. Профіль користувача.
6. KYC процес.
    6.1. Підтвердження телефону.
    6.2. Заповнення профілю.
    6.3. Підтвердження особистості.
7. Сторінка історії запитів на оренду.

Найперше, що користувач бачить на будь-якій платформі, це сторінки реєстрації та авторизації. Кожне поле підсвічується відповідним кольором під час використання, а запит на створення сесії/користувача супроводжується підказкою, що сповіщає користувача про помилку, якщо така є, або про успішне виконання. Дизайн цих сторінок зображено на рис. 4.8.

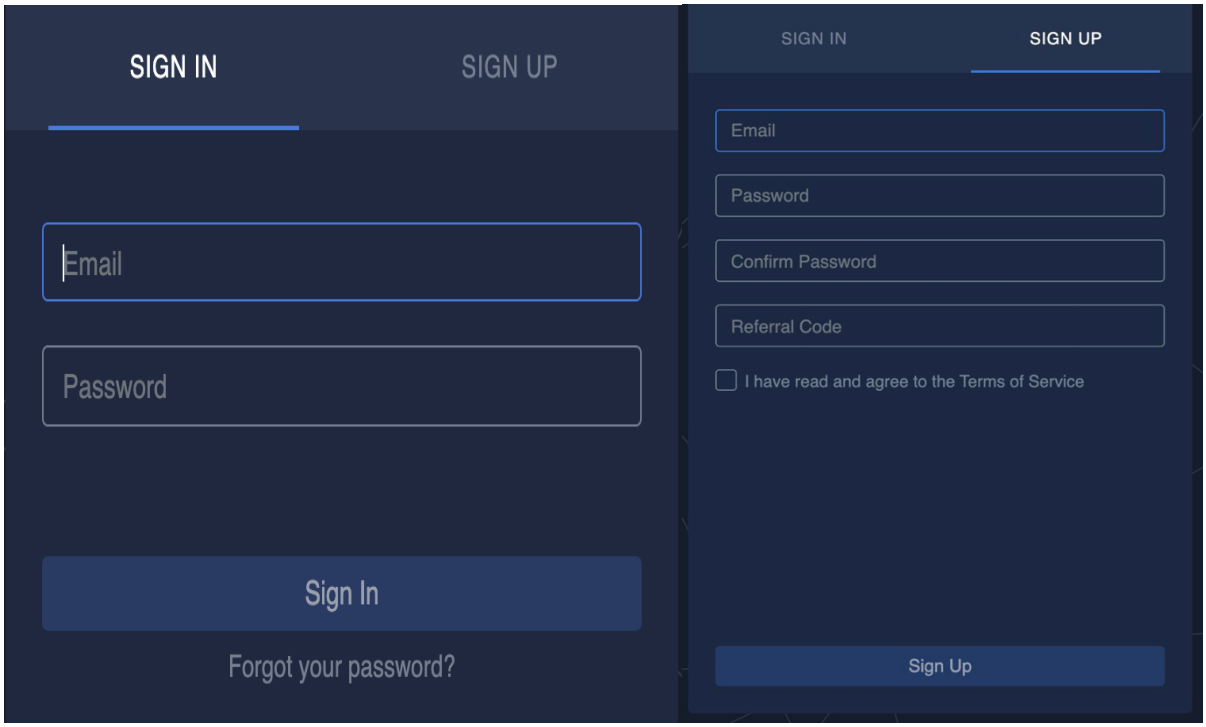


Рис. 4.8. Сторінки авторизації та реєстрації

Будь-який користувач платформи має доступ до власного профілю, де він може переглянути свій поточний рівень КУС, проходження якого повністю обов’язкове лише для орендодавців. Для звичайних користувачів достатньо підтвердити лише електронну пошту. На сторінці свого профілю

користувач також має доступ до інформації про поточний баланс та ETH адрес, таблиці активності, де він може переглянути список всіх авторизацій свого акаунту, IP адресу, браузер та час, коли був зроблений запит на авторизацію, тому користувач може періодично перевіряти, чи не було підозрілої активності в його акаунті. Ця інформація надається та зберігається платформою і не може бути змінена користувачем. Візуальну реалізацію профілю зображено на рис. 4.9.

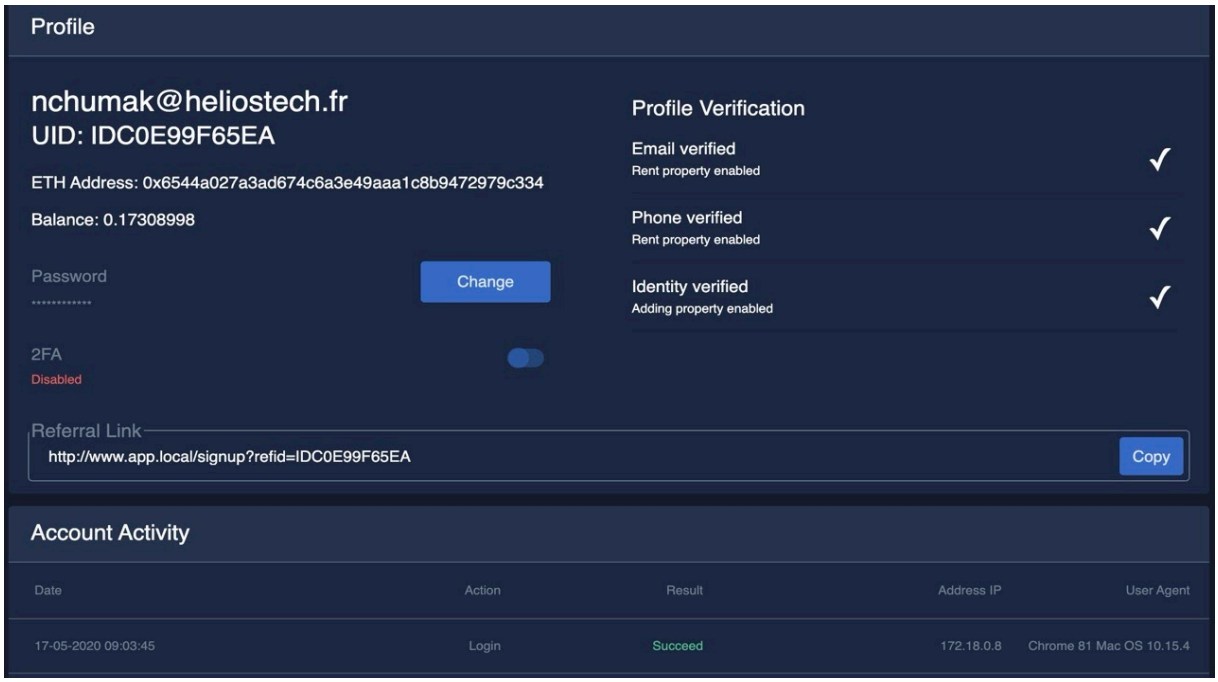


Рис. 4.9. Профіль користувача

Однією з найважливіших сторінок користувацької частини є пошук та перегляд об’єктів оренди, адже це і є основною функціональністю платформи. Вона представлена двома сторінками. Перша, більш загальна, включає в себе динамічний пошук та фільтрацію наявних на сайті об’єктів (рис. 4.10).

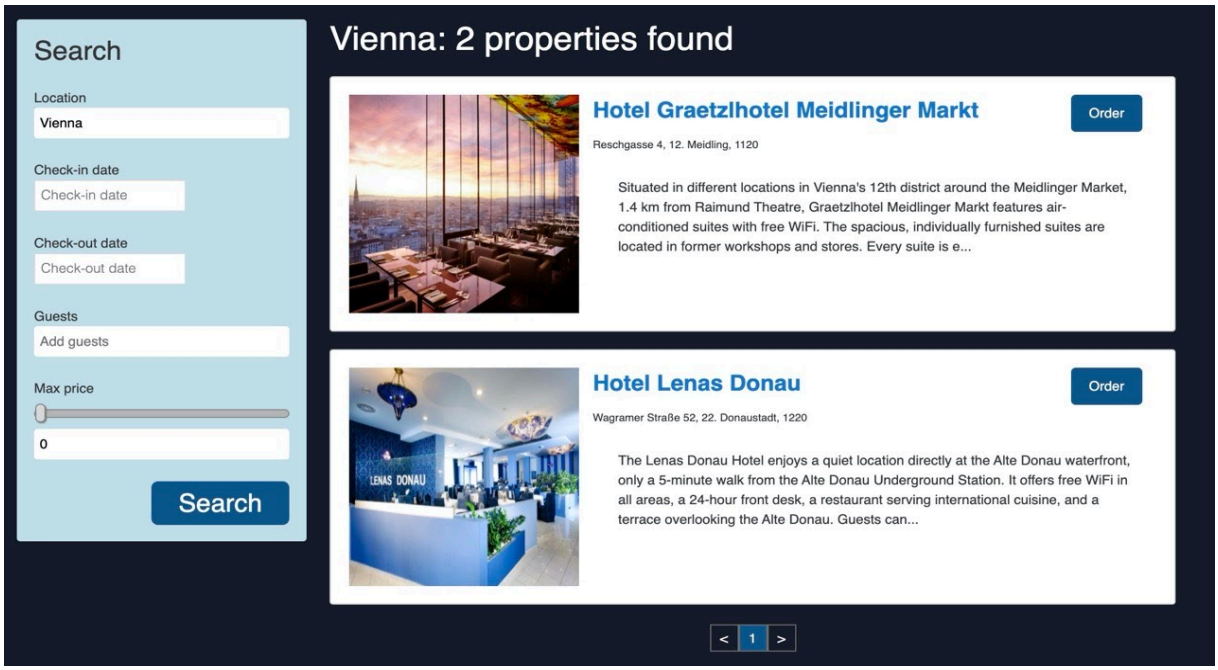


Рис. 4.10. Дизайн сторінки пошуку об’єктів оренди

Користувач може відсортувати за ціновим діапазоном, наявністю вільних місць на конкретні дати, за містом чи адресою, а також за кількістю людей для розміщення. Результат буде показано на цій самій сторінці без перезавантаження. Дана сторінка використовує пагінацію, тому користувач не отримає весь результат відразу, але може переглянути його, перемикаючись між відповідними номерами сторінок результату пошуку.

При натисненні на кнопку “Order”, що висвітлюється біля кожного з знайдених за заданими параметрами результатами, користувач переходить на сторінку з детальною інформацією про об’єкт. Тут у користувача є можливість подивитись опис об’єкту, переглянути наявні об’єкти розміщення, їх ціну, та переглянути фото та медіа-матеріали надані власником (рис. 4.11). Навпроти кожного з об’єктів розміщення наявна кнопка “Reserve”, яка викликає модальне вікно підтвердження бронювання.

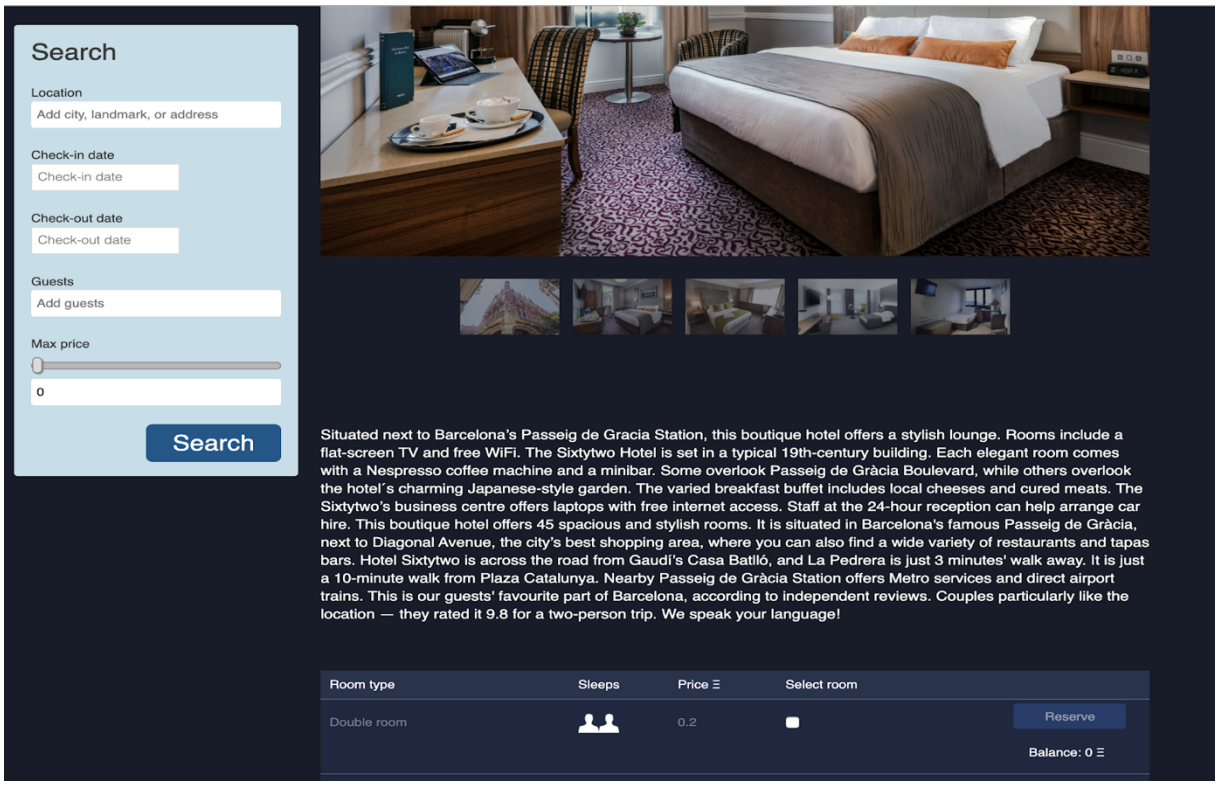


Рис. 4.11. Дизайн сторінки інформації об’єкту оренди

Після підтвердження кошти будуть зняті з рахунку користувача та відповідний смарт-контракт між користувачем та власником буде створений. Якщо коштів не вистачає – кнопка Reserve буде заблокована, а нижче буде вказаний поточний баланс користувача та вибрана кількість днів для оренди.

Кожен користувач має можливість переглянути список своїх бронювань, відмінити бронювання (важливо, що відміняти бронювання можна не менш, ніж за тиждень) або переглянути вже завершені. Сторінка реалізована у вигляді таблиці, що містить основну інформацію про бронювання, ціну бронювання, назву вибраного об’єкту бронювання та об’єкту розміщення, час початку оренди та час закінчення, час створення бронювання та його статус. Сторінка має два підмодулі, де показана історія усіх бронювань та лише активних. Дизайн сторінки зображено на рис. 4.12.

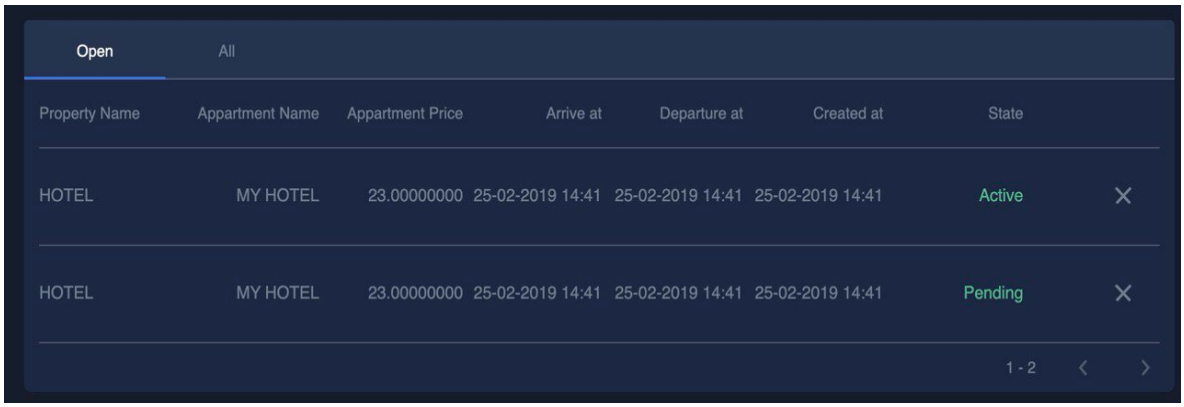


Рис. 4.12. Історія замовлень користувача

4.3. Особливості тестування

Згідно вимог, висунутих до сервісу (детально описані в підрозділі 3.1), було спроєктовано такий набір базових тестів (табл. 4.13), які дозволять протестувати основну функціональність на наявність невідповідностей, помилок конфігурацій, що відповідає техніці димного тестування. Техніка, в більшості випадків, використовується самими розробниками, як стадія початкового тестування програмного забезпечення.

Таблиця 4.13

Тестові випадки

№	Мета тесту	Опис дій тесту	Результат
1	Перевірити наявність та коректну роботу інтерфейсу.	Увімкнути режим “Incognito” у бразері. Перейти на http://www.app.local/profile.	Показується сторінка з формою авторизації, полями вводу електронної адреси і паролю.
2	Перевірити роботу бази даних, e-mail сповіщень.	Увімкнути режим “Incognito” у бразері. Перейти на http://www.app.local/signup. Ввести дані паролю та електронної адреси.	Показується сповіщення про успішну реєстрацію. На пошту приходить лист з підтвердженням.

Продовження табл. 4.13

№	Мета тесту	Опис дій тесту	Результат
3	Перевірити роботу поділу на ролі.	Увімкнути режим “Incognito” у бразері. Авторизуватись як користувач з роллю “member”. Перейти на сторінку http://www.app.local/tower/ users/user-directory.	Показується стандартна початкова сторінка користувацького інтерфейсу.
4	Перевірити наявність та коректну роботу інтерфейсу адмін панелі.	Увімкнути режим “Incognito” у бразері. Авторизуватись як користувач з роллю “admin”. Перейти на сторінку http://www.app.local/tower/ users/user-directory.	Показується сторінка адмін панелі з таблицею списку користувачів платформи.
5	Перевірити роботу створення бронювання, смарт-контракту.	Увімкнути режим “Incognito” у бразері. Авторизуватись як користувач з роллю “member” та з необхідним для бронювання балансом. Перейти на сторінку будь-якого з бронювань, натиснути “Reserve”.	Показується сповіщення про успішне створення бронювання, баланс користувача зменшується на суму бронювання. В історії бронювань з’являється новий об’єкт.
6	Перевірити роботу підтвердження заїзду, зміни статусу смарт-контракту.	Увімкнути режим “Incognito” у бразері. Авторизуватись як користувач з роллю “landlord”, що має активне бронювання. Перейти на сторінку списку бронювань в адмін панелі, ввести пароль від бронювання.	Показується сторінка з таблицею бронювань, бронювання змінило свій статус.
7	Перевірити роботу DB, історії бронювань.	Увімкнути режим “Incognito” у бразері. Авторизуватись як користувач з роллю “member” та існуючими бронюваннями. Перейти на сторінку з історією бронювань.	Показується сторінка з таблицею, кожен рядок якої містить інформацію про унікальне бронювання.

4.4. Рекомендації щодо подальшого використання

Для найбільш ефективного та комфортного використання сервісу, користувач повинен мати базові знання про криптовалюти та блокчейн, мати електронний гаманець і певні кошти на ньому, які йому будуть потрібні при використанні платформи. Особливу увагу кожному користувачу варто приділяти захисту інформації свого електронного гаманця (приватного ключа, паролю).

Також наявна документація та керівництво, що містить відповіді на більшість запитань, які можуть виникнути у нового користувача (як розмістити бронювання, як подивитись статус бронювання, який процес відміни бронювання та повернення коштів, як почати співпрацювати з платформою, як розмістити об’єкт бронювання у власності та інші).

Як для орендарів, так і для орендодавців важливо слідкувати за поточним курсом криптовалюти ETH відповідно до фіатних валют, так як ціни на платформі вказуються саме в ethereum.

Для досвідчених користувачів або розробників для інтеграції у власні сервіси чи використання точок доступу API без інтерфейсу наявна документація з описом наявних API, їх тип, призначення, потрібних для запиту даних, даних, що повертаються, або списку можливих помилок.

ВИСНОВКИ

Метою даного дипломного проєкту було створення програмної системи укладення договорів та домовленостей оренди, де єдиним посередником є розумний контракт, реалізований за допомогою технології блокчейн.

Виконаний у дипломному проєкті аналіз існуючого ринку потреб та попиту користувачів в описаній вище сфері вказав на доцільність створення програми укладення угод оренди, адже дана ніша ринку ще не є перенасиченою постачальниками послуг, а отже, має потенціал для розвитку.

Розроблений веб-сервіс:

1. Дозволяє сплачувати орендні витрати криптомонетою ETH.
2. Зберігає відомості про угоду та її сторони в смарт-контракті, що забезпечує стабільність роботи та незнищенність інформації.
3. Контролює процес угоди алгоритмом смарт-контракту, що гарантує виконання обов’язків оплати та надання оренди.
4. Дозволяє створювати приватний ключ і цифровий підпис для гарантування надійності та непорушності угоди.
5. Перевіряє власників об’єктів оренди згідно KYC та зберігає дані згідно міжнародних стандартів GDPR.

Веб-сервіс реалізовано згідно запланованих вимог (функціональних і нефункціональних), тестування проведено відповідно до затвердженої технології тестування, проведено аналіз відповідності виконаної роботи і поставленого технічного завдання.

Дане програмне забезпечення створить комфортні умови для мандрівок та ділових поїздок, як в міжнародному так і локальному масштабі, що дозволить потенційно збільшити і кількість, і якість укладених угод оренди.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

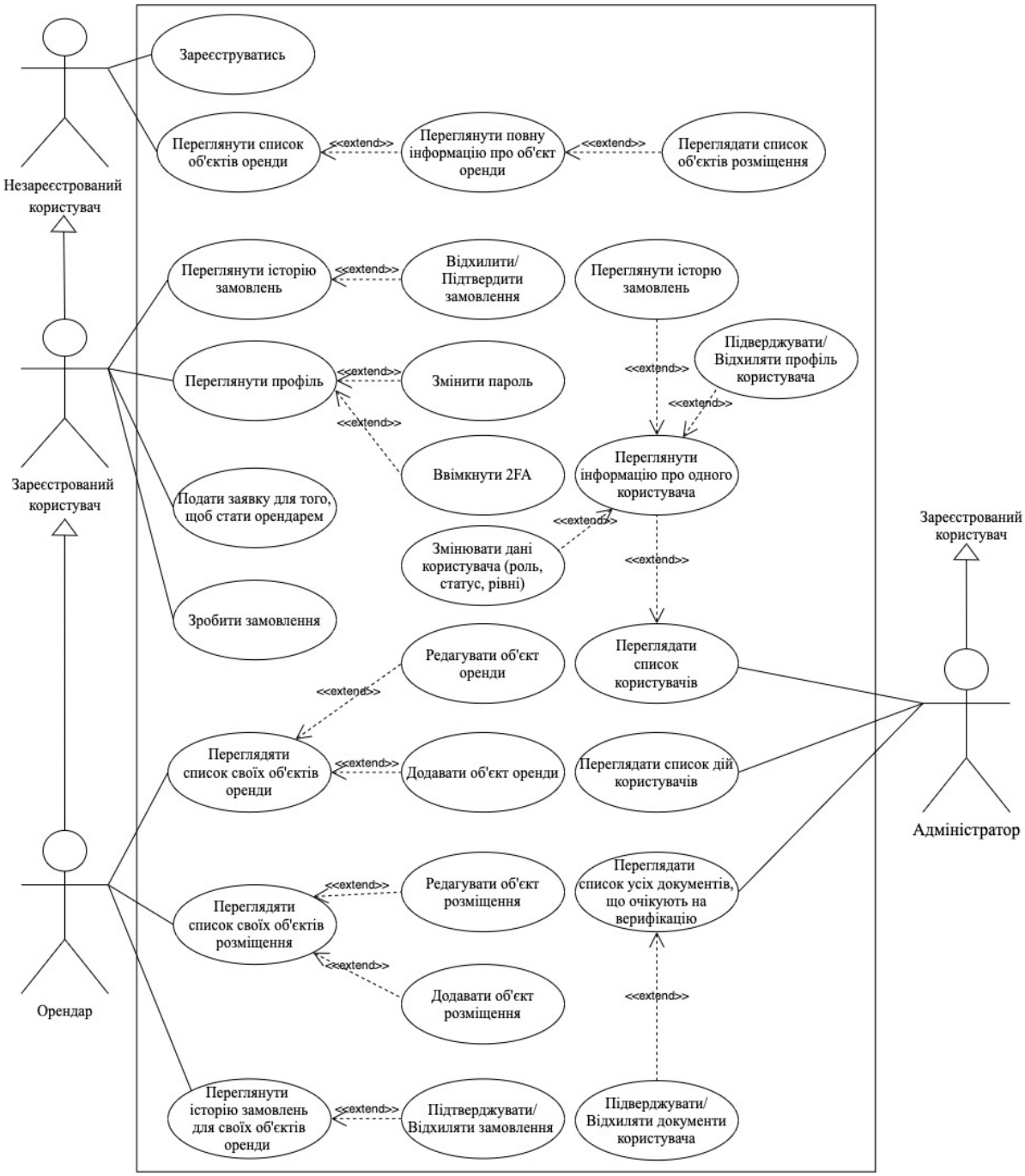
1. Договір оренди: Загальні положення [Електронний ресурс]. – 2014. – Режим доступу: https://protocol.ua/ua/dogovir_orendi_zagalni_pologennya.
2. Цивільний кодекс України від 16.01.2003 р. № 435-IV [Електронний ресурс] // Відомості Верховної Ради України. − 2003. − № 40-44 − ст. 763. − Режим доступу: http://zakon2.rada.gov.ua/laws/show/435-15.
3. Генкин, А. Блокчейн: Как это работает и что ждет нас завтра [Текст] / А. Генкин, А. Михеев. − Альпина Паблишер, 2018. − 9 с.
4. Rentberry – Market your property to millions [Електронний ресурс]. – Режим доступу: https://rentberry.com.
5. Bitsrent – Project Roadmap [Електронний ресурс]. – Режим доступу: https://bitsrent.com.
6. Staybit – Vacation Rentals On the Blockchain [Електронний ресурс]. – Режим доступу: https://staybit.io.
7. How does Booking.com work for property owners [Електронний ресурс]. – Режим доступу: https://partner.booking.com/en-gb/help/working-booking/how-does-bookingcom-work-property-owners.
8. 7 критических преимуществ блокчейна [Електронний ресурс]. – 2016. – Режим доступу: https://bitnovosti.com/2016/05/19/seven-blockchain-benefits-according-to-don-tapscott.
9. Educba – Advantages of Web Service [Електронний ресурс]. – Режим доступу: https://www.educba.com/advantages-of-web-service.
10. Advantages and Disadvantages of web service [Електронний ресурс]. – Режим доступу: http://www.r4r.co.in/webservice/webservice_tutorial/advantages_and_disadvantages_of_web_services.shtml.

11. 3 types of mobile applications: the advantages and disadvantages you should be aware of [Електронний ресурс]. – 2016. – Режим доступу: https://www.genexus.com/en/news/read-news/3-types-of-mobile-applications-the-advantages-and-disadvantages-you-should-be-aware-of.
12. Мартин, Р.А. Идеальный программист. Как стать профессионалом разработки ПО [Текст] / Р.А. Мартин. − Питер, 2012. − 112 с.
13. Introducing Stack Overflow Trends [Електронний ресурс]. – Режим доступу: https://stackoverflow.blog/2017/05/09/introducing-stack-overflow-trends.
14. Stack Overflow [Електронний ресурс]. – Режим доступу: https://en. wikipedia.org/wiki/Stack_Overflow.
15. 10 best Node.js app examples for enterprises, with metrics [Електронний ресурс]. – Режим доступу: https://thinkmobiles.com/blog/node-js-app-examples.
16. Top 57 Ruby Gems in 2020 [Електронний ресурс]. – 2020. – Режим доступу: https://rubygarage.org/blog/best-ruby-gems-we-use.
17. Лучшие языки программирования для бэкенда веб-разработки [Електронний ресурс]. – 2019. – Режим доступу: https://techrocks.ru/2019/01/04/best-backend-web-dev-programming-languages.
18. Фернандес О. Путь Rails. Подробное руководство по созданию приложений в среде Ruby on Rails [Текст] / О. Фернандес. – Символ-Плюс, 2009. − 202 с.
19. Frontend - как все поменялось за последние 15 лет [Електронний ресурс]. – Режим доступу: https://xakep.ru/2014/05/30/frontend-changes.
20. 2019 Stats on Top JS Frameworks [Електронний ресурс]. – Режим доступу: https://www.tecla.io/blog/2019-stats-on-top-js-frameworks-react-angular-and-vue.
21. Сможет ли Vue.js превзойти React в 2020 году? [Електронний ресурс]. – Режим доступу: https://nuancesprog.ru/p/7064.

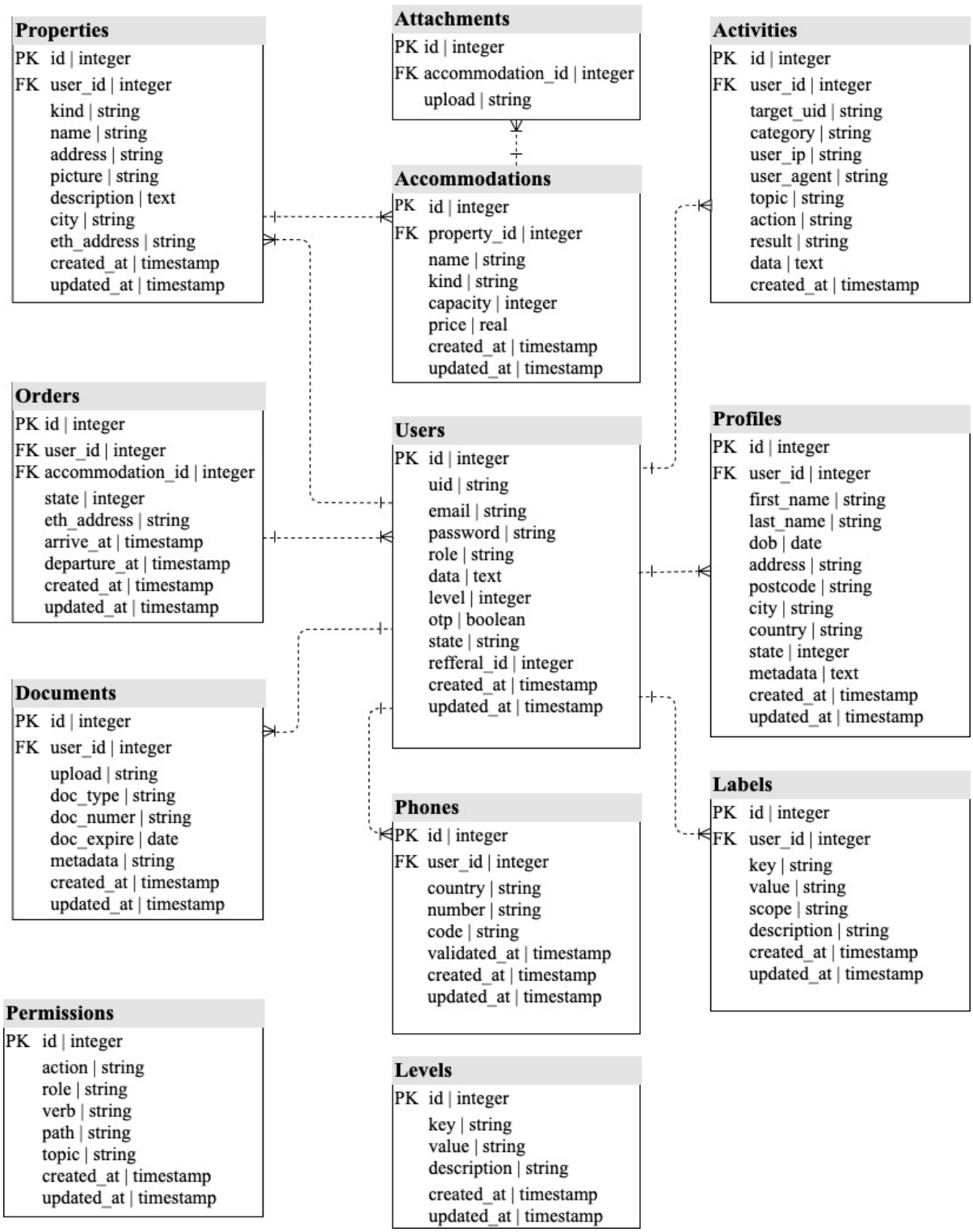
22. React repository on github [Електронний ресурс]. – Режим доступу: https://github.com/facebook/react.
23. Технология Blockchain. Практическое применение [Електронний ресурс]. – 2016. – Режим доступу: https://blog.dti.team/tekhnologiya-blockchain-prakticheskoe-primenenie.
24. Кардонов, А.В. Сферы применения смарт-контрактов и риски при работе с ними [Текст] / А.В. Кардонов. – Иркутск, 2018. − 44 с.
25. Как создать смарт-контракт для начинающих [Електронний ресурс]. – 2018. – Режим доступу: https://ru.ihodl.com/tutorials/2018-04-23/kak-sozdat-smart-kontrakt-instrukciya-dlya-nachinayushih.
26. Преимущества и недостатки смарт-контрактов [Електронний ресурс]. – Режим доступу: https://polygant.net/ru/blog/preimushhestva-i-nedostatki-smart-kontraktov.
27. Are Blockchain smart contracts all they touted to be?[Електронний ресурс]. – 2019. – Режим доступу: https://www.deccanherald.com/opinion/panorama/are-blockchain-smart-contracts-all-they-touted-to-be-770376.html
28. Mikowski Michael. Single Page Web Applications [Текст] / Michael Mikowski. – Manning, 2013. – 200c. – ISBN 9781617290756.
29. Микросервисы [Електронний ресурс]. – 2015. – Режим доступу: https://habr.com/ru/post/249183.
30. Pattern: API Gateway / Backends for Frontends [Електронний ресурс]. – Режим доступу: https://microservices.io/patterns/apigateway.html.
31. Microservices Authentication and Authorization Solutions [Електронний ресурс]. – Режим доступу: https://medium.com/tech-tajawal/microservice-authentication-and-authorization-solutions-e0e5e74b248a.

ДОДАТКИ

Додаток 1
Копії графічних матеріалів

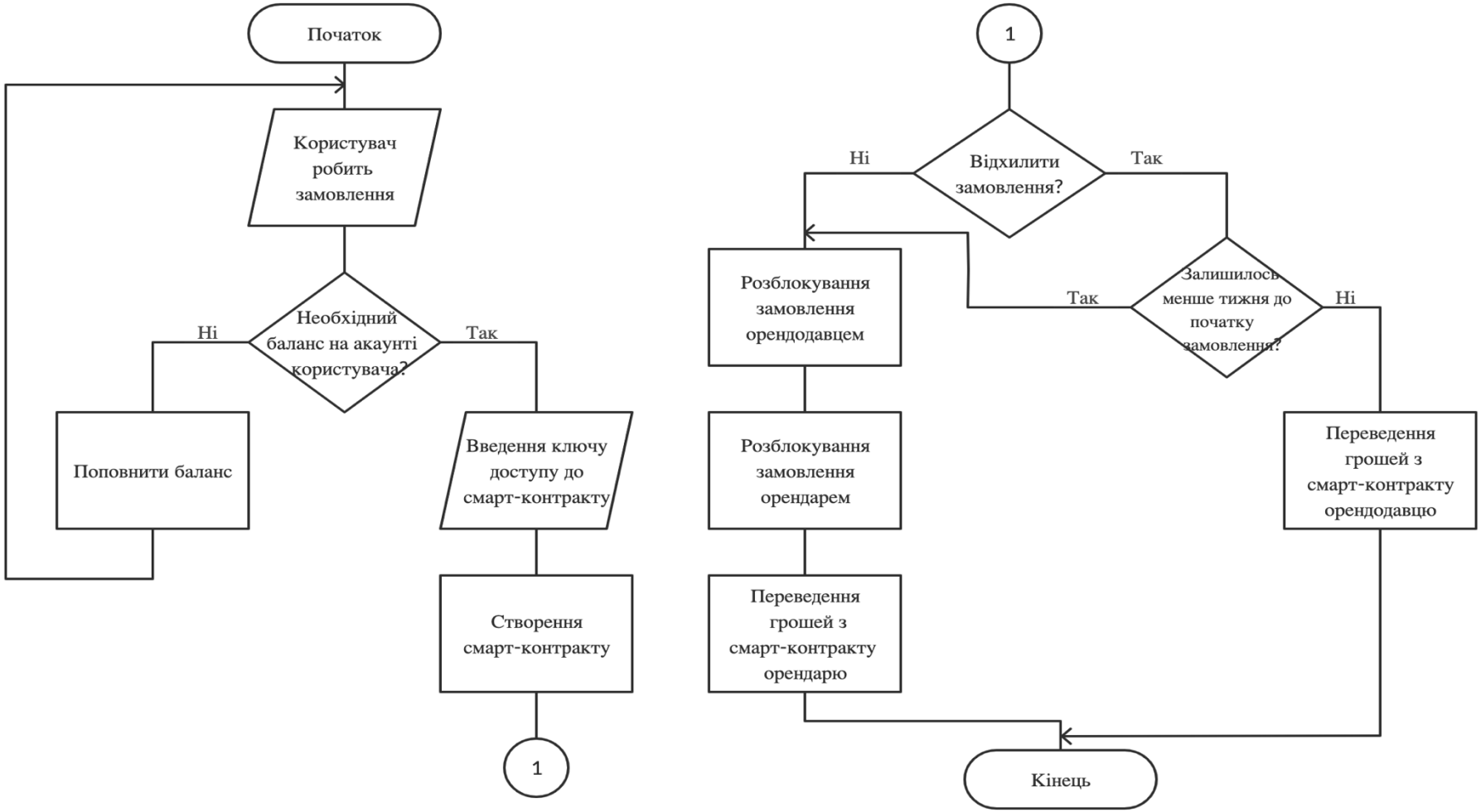


ДП.045440-06-99
Веб-сервіс для укладання безпечних угод оренди на основі технології блокчейн.
Функціональність програмних засобів.
UML-діаграма прецедентів



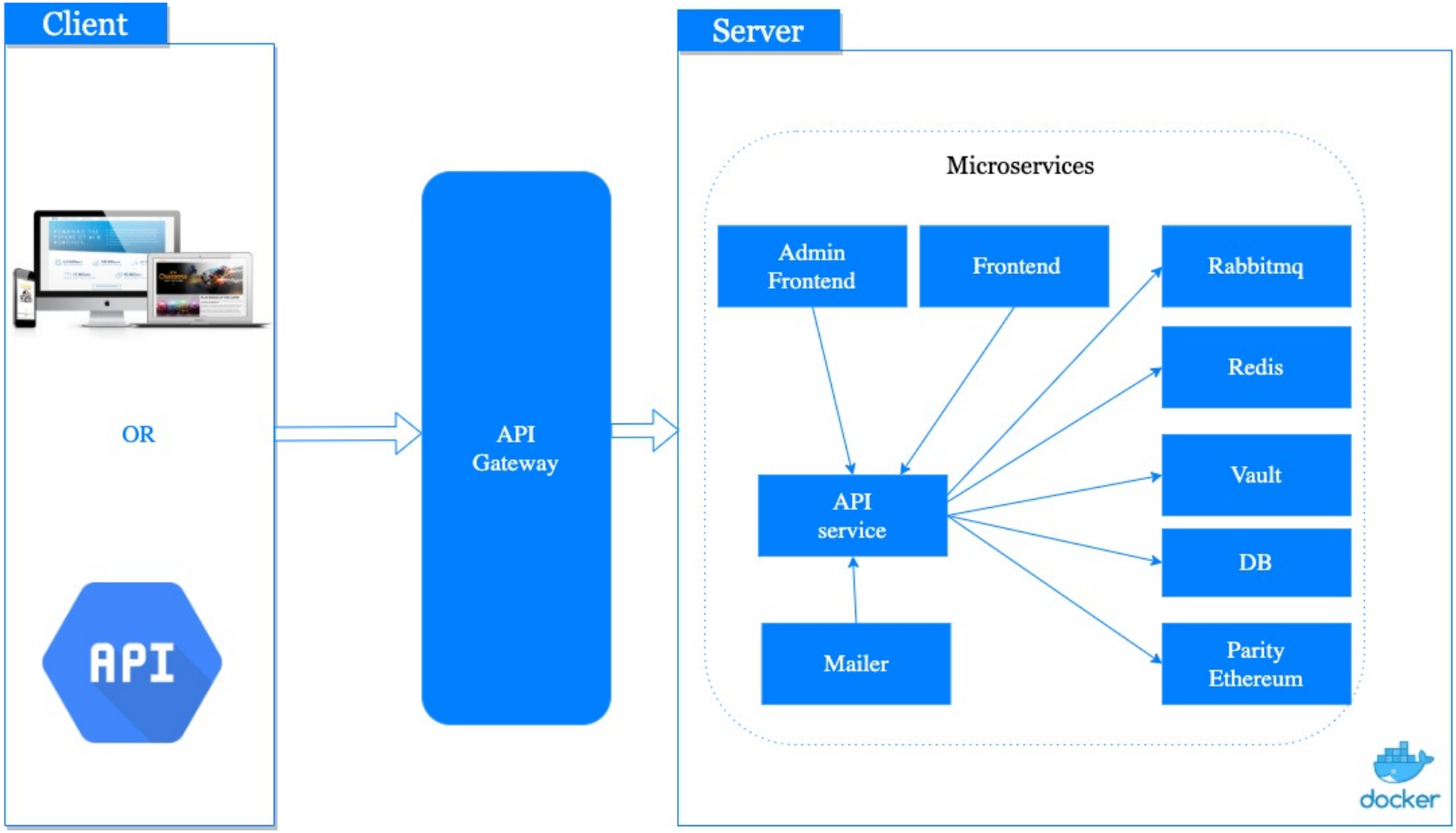
ДП.045440-07-99
Веб-сервіс для укладання безпечних угод оренди на основі технології блокчейн.
Структура бази даних.
ER-діаграма

Схема алгоритму роботи смарт-контракту



Чумак Н.Р., група КП-61

Схема взаємодії програмних модулів



Чумак Н.Р., група КП-61

Додаток 2
Лістинг програми

Лістинг модуля смарт-контракту

```
pragma solidity 0.4.18;
contract RentalAgreement {
    mapping(address => uint256) balances;
    uint public rent;
    string public passphrase;
    uint public createdTimestamp;
    address public landlord;
    address public tenant;
    enum State { Created, Deposited, Started, Payed, Terminated }
    State public state;
    function RentalAgreement(address _landlord, address _tenant, uint
_rent, string _pass) {
        landlord = _landlord;
        tenant = _tenant;
        rent = _rent;
        createdTimestamp = block.timestamp;
        state = State.Created;
        passphrase = _pass;
    }
    modifier require(bool _condition) {
        if (!_condition) throw;
        _;
    }
    modifier onlyLandlord() {
        if (msg.sender != landlord) throw;
         _;
    }
    modifier onlyTenant() {
        if (msg.sender != tenant) throw;
        _;
    }
    modifier inState(State _state) {
        if (state != _state) throw;
        _;
    }
    function getLandlord() public view returns (address) {
        return landlord;
    }
    function getLandlordBalance() public view returns (uint) {
        return balances[landlord];
    }
    function getTenant() public view returns (address) {
        return tenant;
    }
    function getTenantBalance() public view returns (uint) {
        return balances[tenant];
    }
    function getRent() public view returns (uint) {
        return rent;
    }
    function getContractCreated() public view returns (uint) {
        return createdTimestamp;
    }
    function getContractAddress() public view returns (address) {
        return this;
    }
    function getContractBalance() public view returns (uint){
        return address(this).balance;
    }
    function getState() returns (State) {
        return state;
```

```
    }
    function passphraseCheck(string pass) public returns (bool) {
        return keccak256(pass) == keccak256(passphrase);
    }
    function payMoney() {
        balances[landlord] += this.balance;
        landlord.send(this.balance);
        state = State.Payed;
    }
    function deposit() public payable
    onlyTenant
    require (msg.value == rent) {
        balances[msg.sender] += msg.value;
        state = State.Deposited;
    }
    function() public payable {
        deposit();
    }
    function terminateOrder() {
        tenant.send(this.balance);
        balances[msg.sender] -= this.balance;
        state = State.Terminated;
    }
    function changeState() {
        state = State.Started;
    }
}
```

Лістинг API модуля

```
# frozen_string_literal: true

module API
  module V2
    module Resource
      # Responsible for CRUD for orders
      class Orders < Grape::API
        resource :orders do
          helpers ::API::V2::NamedParams
          helpers ::API::V2::Admin::NamedParams

          helpers do
            def accomodation_busy?
              from = params[:arrive_at]
              to = params[:departure_at]

              sql = "(arrive_at >= ? AND departure_at <= ?)
                       OR (arrive_at <= ? AND departure_at >= ?)
                       OR (arrive_at <= ? AND departure_at >= ?)
                       OR (arrive_at <= ? AND departure_at >= ?)"
              return false if @accomodation.orders
                .where('state != ? AND state != ?', '4', '3')
                .where(sql, from, to, from, to, from, from, to, to).empty?

              true
            end
          end

          desc 'List all orders for current user.',
               security: [{ "BearerToken": [] }],
               failure: [
                 { code: 401, message: 'Invalid bearer token' }
```

```
    ]
params do
  optional :state, type: String
  use :pagination_filters
end
get do
  orders = current_user.orders
  orders = orders
             .where("state = 1 OR state = 0") if params[:state]

  present paginate(orders), with: Entities::Order
end

desc 'Unlocks money',
security: [{ "BearerToken": [] }],
failure: [
  { code: 401, message: 'Invalid bearer token' }
]
params do
  requires :passphrase, type: String
  requires :order_id, type: Integer
end
post '/unlock' do
  order = current_user.orders.find_by(id: params[:order_id])
  error!("resource.orders.order_invalid", 422) unless order
  error!("resource.orders.order_not_active", 422) if
                              order.state != 'active' # active
  params[:passphrase] = Digest::SHA2
                              .new(256)
                              .hexdigest(params[:passphrase])

  property = order.accomodation.property
  contract = Ethereum::Contract.create(
             client:Rentware::App.config.eth_client,
             name: "RentalAgreement",
             address: order.eth_address,
             abi: SecretStorage
                 .get_secret('contract_abi').data[:value])

  decrypted_key = Eth::Key
                      .decrypt File.read(SecretStorage
                      .get_secret(
                      "key_path_#{current_user.uid}")
                      .data[:value]),
  SecretStorage
      .get_secret("key_password_#{current_user.uid}").data[:value]

  contract.key = decrypted_key
  contract.sender = current_user.eth_address
  error!("landlord.accomodations.passphrase_invalid", 422)
     unless contract.call.passphrase_check(params[:passphrase])

  contract.transact.pay_money
  error!("resource.orders.payment_failed", 422)
                     if contract.call.get_state != 3

  error!("resource.orders.money_transfer_failed", 422)
             if contract.call.get_contract_balance != 0 &&
                contract.call.get_landlord_balance == 0

  order.update(state: 3)
  status 201
rescue IOError => e
```

```
  error!("resource.orders.insufficient_balance", 422)
              if e.message.include?('Insufficient')
end

desc 'Create new order for current user.',
security: [{ "BearerToken": [] }],
   failure: [
     { code: 400, message: 'Required params are empty' },
     { code: 401, message: 'Invalid bearer token' },
     { code: 422, message: 'Validation errors' }
   ]
params do
  requires :accomodation_id, type: Integer
  requires :arrive_at, type: String
  requires :departure_at, type: String
  requires :passphrase, type: String
end
post do
  @accomodation = Accomodation
                      .find_by(id: params[:accomodation_id])
  error!("resource.orders.accomodation_invalid", 422)
                                    unless @accomodation

  params[:arrive_at] = Time.parse(params[:arrive_at])
  params[:departure_at] = Time.parse(params[:departure_at])
  price = (((@accomodation.price * (params[:departure_at].to_date
  - params[:arrive_at].to_date).round).to_f) * 10**18).to_i

  params[:state] = 0
  params[:passphrase] = Digest::SHA2
                             .new(256)
                             .hexdigest(params[:passphrase])

  error!("resource.orders.accomodation_busy", 422)
                                    if accomodation_busy?

  contract = Ethereum::Contract.create(
            client: Rentware::App.config.eth_client, file:
            Rails.root.join("rental_agreement.sol"))

  params[:eth_address] = contract
                          .deploy_and_wait(
                             @accomodation.property.eth_address,
                              current_user.eth_address,
                              price, params[:passphrase] )

  order = current_user.orders.new(params.except(:passphrase))
  code_error!(order.errors.details, 422) unless order.save

  SecretStorage.store_secret(contract.abi, 'contract_abi')

  decrypted_key = Eth::Key
                      .decrypt File.read(SecretStorage
                      .get_secret("key_path_#{current_user.uid}")
                      .data[:value]), SecretStorage
     .get_secret("key_password_#{current_user.uid}").data[:value]
  contract.key = decrypted_key

  contract.parent
     .encode_tx({ value: price },
      contract.parent.functions[15])

  contract.sender = current_user.eth_address
```

```
  EventAPI.notify('system.user.order.created',
  record: {
    order: order.as_json_for_event_api,
    domain: Rentware::App.config.domain
  })
  present order, with: API::V2::Entities::Order
  status 201
rescue IOError => e
  order.delete
  error!("resource.orders.insufficient_balance", 422)
                    if e.message.include?('Insufficient')
end

desc 'Cancel order for current user.',
security: [{ "BearerToken": [] }],
   failure: [
     { code: 400, message: 'Required params are empty' },
     { code: 401, message: 'Invalid bearer token' },
     { code: 422, message: 'Validation errors' }
   ]
params do
  requires :id, type: Integer
end
post '/:id' do
  order = current_user.orders.find_by(id: params[:id])
  error!("resource.orders.order_invalid", 422) unless order

  error!("resource.orders.cant_cancel_order", 422)
     if (order.arrive_at.to_date - Time.now.to_date).to_i < 7

  # retrieve contract here
  contract = Ethereum::Contract.create(
             client: Rentware::App.config.eth_client,
             name: "RentalAgreement",
             address: order.eth_address,
             abi:SecretStorage
                     .get_secret('contract_abi').data[:value])
  decrypted_key = Eth::Key
                    .decrypt
                    File.read(SecretStorage.get_secret("key_path_
                    #{current_user.uid}").data[:value]),
                    SecretStorage.get_secret("key_password_#{curr
                    ent_user.uid}").data[:value]
  contract.key = decrypted_key
  contract.transact.terminate_order
  contract.sender = current_user.eth_address

  # if some money left on contract or rent not
  # cleared properly - deny order cancel
  error!("resource.orders.cancel_failed", 422)
                            if contract.call.get_state != 4
  error!("resource.orders.money_refund_failed")
     if contract.call.get_contract_balance != 0 &&
         contract.call.get_tenant_balance != 0

  order.update(state: 4) # rejected
  EventAPI.notify('system.user.order.rejected',
  record: {
    order: order.as_json_for_event_api,
    domain: Rentware::App.config.domain
  })
```

```
            present order, with: API::V2::Entities::Order
            status 201
          rescue IOError => e
            error!("resource.orders.insufficient_balance", 422) if
e.message.include?('Insufficient')
          end
        end
      end
    end
  end
end

# frozen_string_literal: true

class Order < ApplicationRecord
  belongs_to :user
  belongs_to :accomodation

  validate :dates_valid!

  enum state: { pending: 0, active: 1, confirmed: 2, finished: 3, rejected:
4 }

  def dates_valid!
    return if arrive_at.blank? || departure_at.blank?

    if arrive_at.past?
      errors.add(:base, 'Arrival date can not be in past')
    elsif departure_at.past?
      errors.add(:base, 'Departure date can not be in past')
    elsif departure_at < arrive_at
      errors.add(:base, 'Departure date can not be earlier than arrival
date')
    end
  end

  def as_json_for_event_api
    {
      user: user.as_json_for_event_api,
      accomodation: accomodation.as_json_for_event_api,
      state: state,
      arrive_at: format_iso8601_time(arrive_at),
      departure_at: format_iso8601_time(departure_at),
      created_at: format_iso8601_time(created_at),
      updated_at: format_iso8601_time(updated_at)
    }
  end
end

# frozen_string_literal: true

require 'active_support/configurable'

class Accomodation < ApplicationRecord
  include ActiveSupport::Configurable

  belongs_to :property
  has_many :attachments
  has_many :orders

  validates :name, :kind, :capacity, :price,
             presence: {
```

```
                message: ->(_, data) {
"missing_#{data[:attribute].parameterize(separator: '_')}" }
              }
  validates :price, :capacity,
            numericality: {
              greater_than: 0,
              message:      ->(_, data) {
"non_positive_#{data[:attribute].parameterize(separator: '_')}" }
            }

  def as_json_for_event_api
    {
      property: property.as_json_for_event_api,
      name: name,
      kind: kind,
      capacity: capacity,
      price: price,
      created_at: format_iso8601_time(created_at),
      updated_at: format_iso8601_time(updated_at)
    }
  end
end

# frozen_string_literal: true

class Property < ApplicationRecord
  mount_uploader :picture, Rentware::App.config.uploader

  validates :name, presence: true, uniqueness: true

  has_many :accomodations
  belongs_to :user

  def as_json_for_event_api
    {
      name: name,
      kind: kind,
      address: address,
      description: description,
    }
  end
end

# frozen_string_literal: true

# User model
class User < ApplicationRecord
  acts_as_eventable prefix: 'user', on: %i[create update]
  has_secure_password

  has_many  :orders,        dependent: :destroy
  has_many  :profiles,      dependent: :destroy
  has_many  :phones,        dependent: :destroy
  has_many  :data_storages, dependent: :destroy
  has_many  :documents,     dependent: :destroy
  has_many  :labels,        dependent: :destroy
  has_many  :activities,    dependent: :destroy
  has_many  :properties,    dependent: :destroy
  has_many  :accomodations, :through => :properties
  # has_many  :orders, :through => :accomodations

  validates_length_of :data, maximum: 1024
  validate :role_exists
```

```
  validate :referral_exists
  validates :data, data_is_json: true
  validates :email,       email: true, presence: true, uniqueness: true
  validates :uid,         presence: true, uniqueness: true
  validates :password,    presence: true, if: :should_validate?
  validate  :validate_pass!

  scope :active, -> { where(state: 'active') }
  scope :with_pending_or_replaced_docs, -> {
self.joins(:labels).where(labels:
                                          { key: 'document', value:
['pending', 'replaced'], scope: 'private' }) }

  before_validation :assign_uid
  after_update :disable_api_keys

  def validate_pass!
    return unless (new_record? && password.present?) || password.present?

    validation_result = PasswordStrengthChecker.validate!(password)
    errors.add(:password, validation_result) unless validation_result ==
'strong'
  end

  def disable_api_keys
    if otp_previously_changed? && otp == false || state_previously_changed?
&& state != 'active'
      api_keys.active.each do |key|
        key.update(state: 'inactive')
      end
    end
  end

  def active?
    self.state == 'active'
  end

  def superadmin?
    self.role == 'superadmin'
  end

  def role_exists
    return if Permission.pluck(:role).include?(role)

    errors.add(:role, 'doesnt_exist')
  end

  # Check if refferal exist for assignment
  def referral_exists
    errors.add(:referral_id, 'doesnt_exist') if referral_id.present? &&
User.find_by(id: referral_id).blank?
  end

  def referral_uid
    user = User.find_by(id: referral_id)
    return if user.nil?

    user.uid
  end

  def role
    super.inquiry
  end
```

```
  def should_validate?
    new_record? || password.present?
  end

  # FIXME: Clean level micro code
  def update_level
    user_level = 0
    tags = labels.with_private_scope
                 .map { |l| [l.key, l.value].join ':' }

    levels = Level.all.order(id: :asc)
    levels.each do |lvl|
      break unless tags.include?(lvl.key + ':' + lvl.value)

      user_level = lvl.id
    end
    update(level: user_level)
  end

  def update_state
    @resulting_state = 'pending'

    # check if user has all required labels for activation
    if labels_include?(RentwareConfig.list['activation_requirements'])
      @resulting_state = 'active'
    end
    RenwareConfig.list['state_triggers']&.each do |state, triggers|
      triggers.each { |trigger|
        labels.pluck(:key).each { |label|
          @resulting_state = state if label.start_with?(trigger)
        }
      }
    end

    update(state: @resulting_state) if @resulting_state != self.state
  end

  # check if given key: values hash is a subset of private user labels
  def labels_include?(labels_hash)
    labels_hash <= private_labels_to_hash
  end

  # Select all key-value pairs from user labels with private scope, merge
in one hash
  def private_labels_to_hash
    key_value_arr = self.labels.with_private_scope.map do
      |l| { l.key => l.value }
    end
    key_value_hash = key_value_arr.inject(:merge)
    key_value_hash || {}
  end

  def as_json_for_event_api
    {
      uid: uid,
      email: email,
      role: role,
      level: level,
      otp: otp,
      state: state,
      referral_uid: referral_uid,
      created_at: format_iso8601_time(created_at),
```

```
      updated_at: format_iso8601_time(updated_at)
    }
  end

  def as_payload
    as_json(only: %i[uid email referral_id role level state])
  end
  def language
    if data.blank?
      Rentware::App.config.default_language
    else
      JSON.parse(data)['language'] ||Rentware::App.config.default_language
    end
  end

  def submitted_profile
    self.profiles&.find_by(state: 'submitted')
  end

  def drafted_profile
    self.profiles&.find_by(state: 'drafted')
  end

  private

  def assign_uid
    return unless uid.blank?

    loop do
      self.uid = random_uid
      break unless User.where(uid: uid).any?
    end
  end

  def random_uid
    "%s%s" % [Renware::App.config.uid_prefix.upcase,
SecureRandom.hex(5).upcase]
  end
end

# Data Storage model
class DataStorage < ApplicationRecord
  BLACKLISTED_TITLES = %w[document label profile phone user].freeze
  acts_as_eventable prefix: 'data_storage', on: %i[create update]
  belongs_to :user
  validates :title, :data, presence: true
  validates_length_of :data, maximum: 5120 # maximum 5kb of data
  validates :data, data_is_json: true
  validates :title, uniqueness: { scope: :user_id, case_sensitive: false },
                    inclusion: { in: UserStorageTitles.list }, exclusion: {
in: BLACKLISTED_TITLES }
  def as_json_for_event_api
    {
      user: user.as_json_for_event_api,
      title: title,
      data: data,
      created_at: format_iso8601_time(created_at),
      updated_at: format_iso8601_time(updated_at)
    }
  end
end
```

Додаток 3
Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ
КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

ВЕБ-СЕРВІС ДЛЯ УКЛАДАННЯ БЕЗПЕЧНИХ УГОД ОРЕНДИ БЕЗ ПОСЕРЕДНИКІВ НА ОСНОВІ ТЕХНОЛОГІЇ BLOCKCHAIN

Виконала: Чумак Надія Русланівна

Керівник: ст. викл. Гадиняк Руслан Анатолійович

Київ – 2020

ПОСТАНОВКА ЗАДАЧІ

Мета дипломного проєкту:

Розробити веб-сервіс для укладання безпечних угод оренди без посередників, використовуючи смарт-контракти та технологію блокчейн.

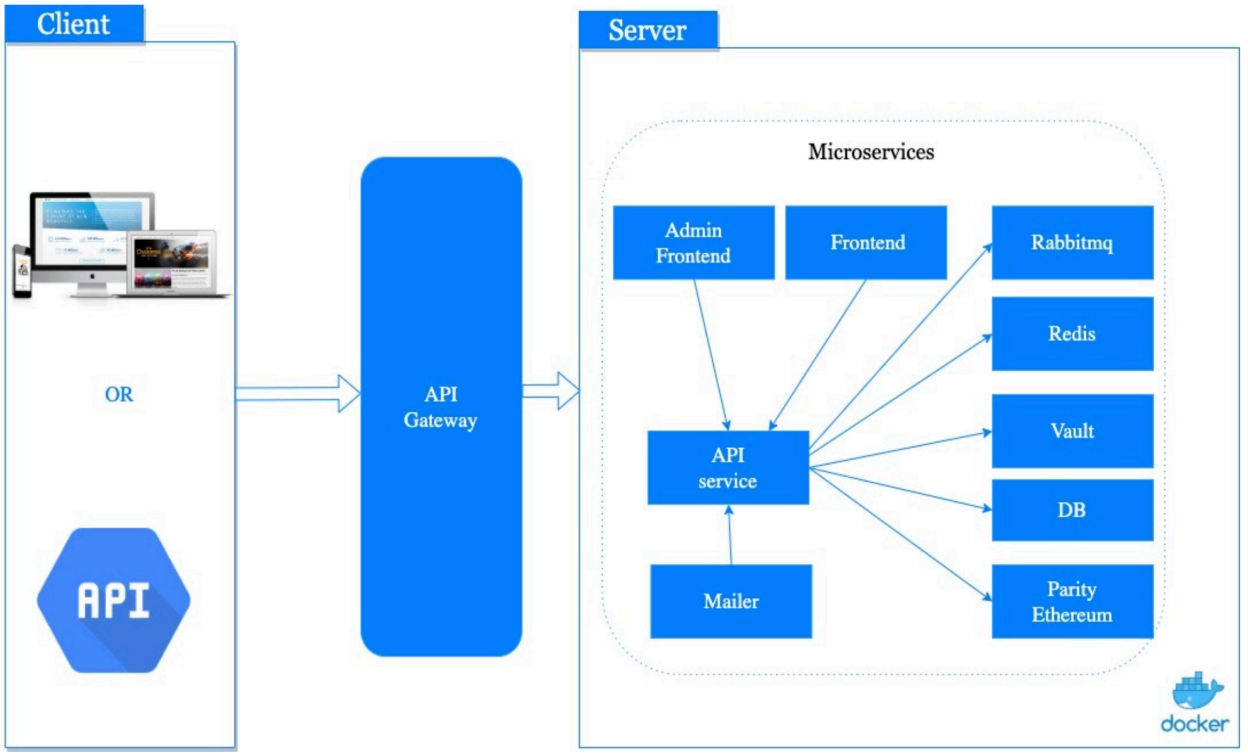
Завдання:

- Проаналізувати наявні системи укладання договорів оренди, їх переваги та недоліки
- Розробити алгоритм смарт-контракту
- Розробити веб-сервіс для укладання угод оренди
- Протестувати розроблений алгоритм смарт-контракту та його інтеграцію в програмну систему

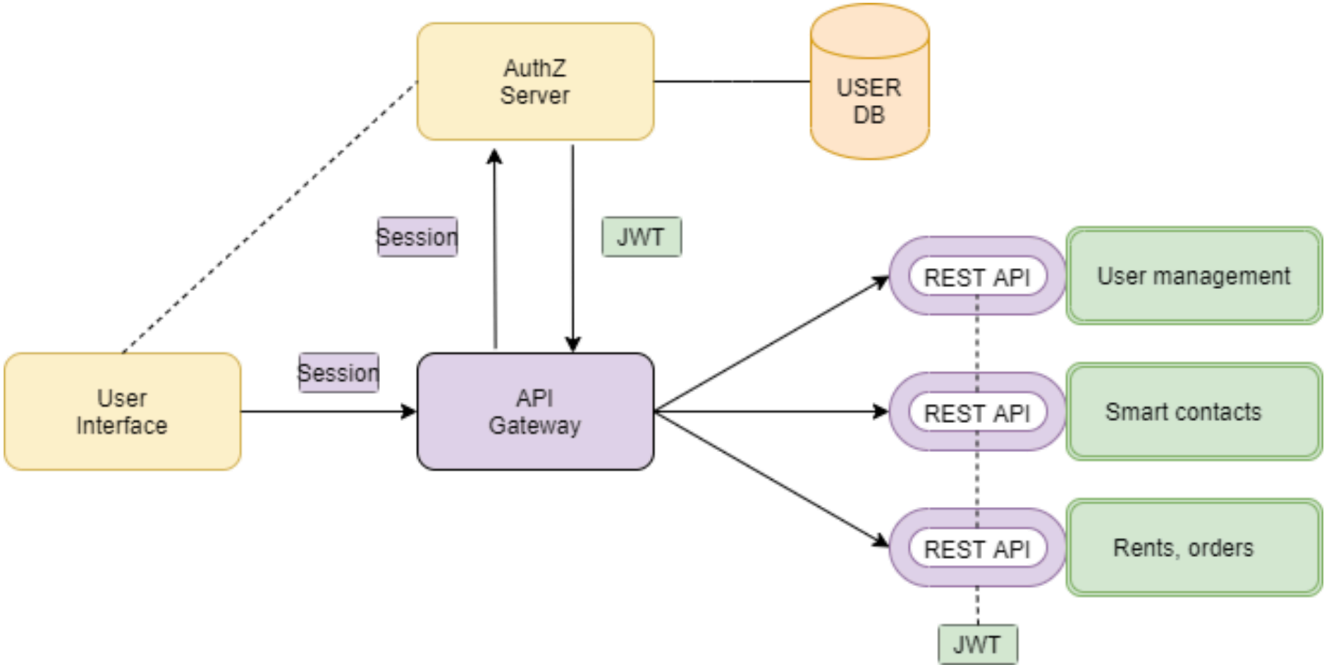
АКТУАЛЬНІСТЬ

- Система, що не потребує ролі посередника
- Смарт-контракт як гарант надійності
- Зменшення часу укладання договорів за рахунок відсутності юридичних підтверджень
- Використання криптовалюти як інтернаціонального способу оплати
- Зменшення комісійних витрат

АРХІТЕКТУРА ТА КОМПОНЕНТИ СИСТЕМИ

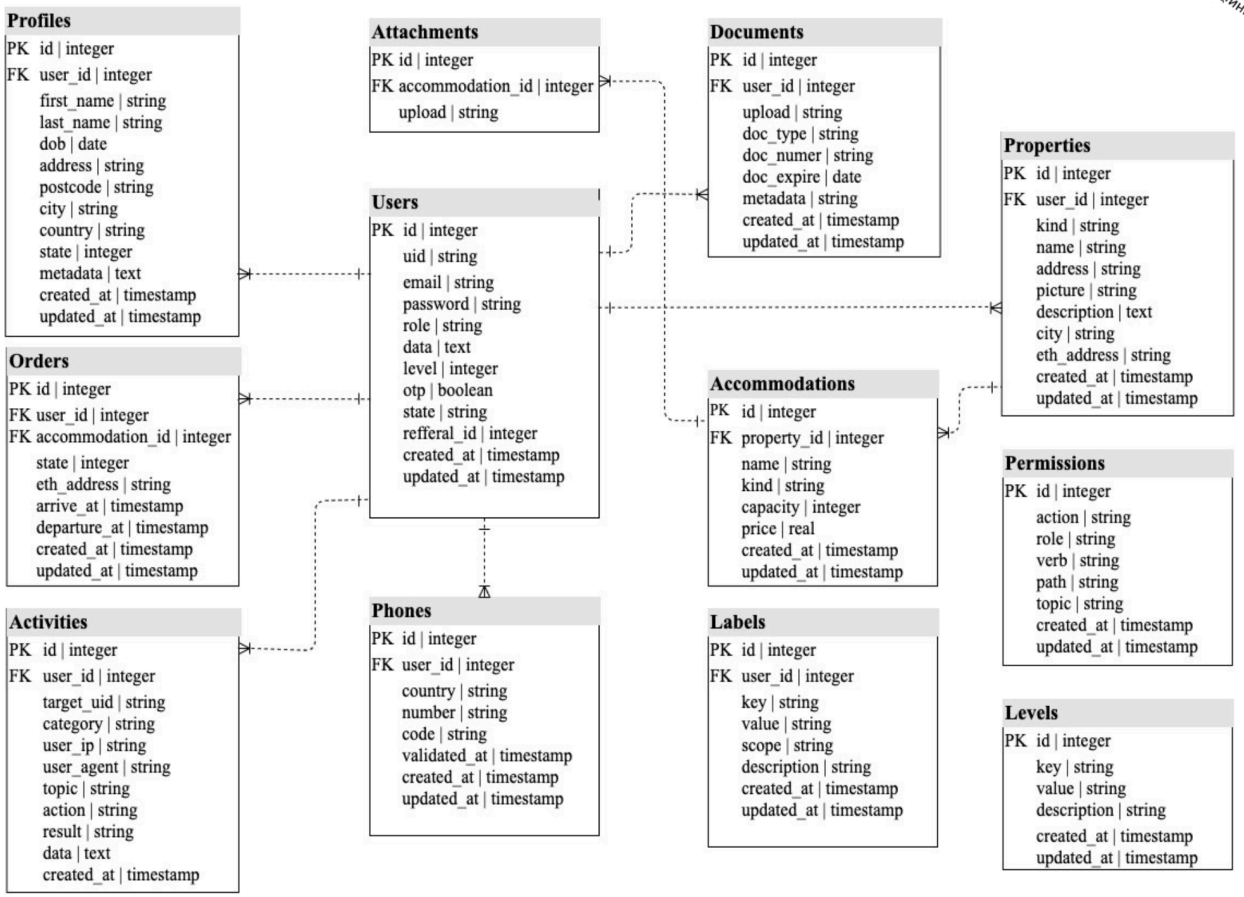


АРХІТЕКТУРА СИСТЕМИ



ЗАСОБИ РЕАЛІЗАЦІЇ

БАЗА ДАНИХ


Profiles
PK id | integer
FK user_id | integer
first_name | string
last_name | string
dob | date
address | string
postcode | string
city | string
country | string
state | integer
metadata | text
created_at | timestamp
updated_at | timestamp
Attachments
PK id | integer
FK accommodation_id | integer
upload | string
Documents
PK id | integer
FK user_id | integer
upload | string
doc_type | string
doc_numer | string
doc_expire | date
metadata | string
created_at | timestamp
updated_at | timestamp
Properties
PK id | integer
FK user_id | integer
kind | string
name | string
address | string
picture | string
description | text
city | string
eth_address | string
created_at | timestamp
updated_at | timestamp
Users
PK id | integer
uid | string
email | string
password | string
role | string
data | text
level | integer
otp | boolean
state | string
refferal_id | integer
created_at | timestamp
updated_at | timestamp
Orders
PK id | integer
FK user_id | integer
FK accommodation_id | integer
state | integer
eth_address | string
arrive_at | timestamp
departure_at | timestamp
created_at | timestamp
updated_at | timestamp
Accommodations
PK id | integer
FK property_id | integer
name | string
kind | string
capacity | integer
price | real
created_at | timestamp
updated_at | timestamp
Permissions
PK id | integer
action | string
role | string
verb | string
path | string
topic | string
created_at | timestamp
updated_at | timestamp
Activities
PK id | integer
FK user_id | integer
target_uid | string
category | string
user_ip | string
user_agent | string
topic | string
action | string
result | string
data | text
created_at | timestamp
Phones
PK id | integer
FK user_id | integer
country | string
number | string
code | string
validated_at | timestamp
created_at | timestamp
updated_at | timestamp
Labels
PK id | integer
FK user_id | integer
key | string
value | string
scope | string
description | string
created_at | timestamp
updated_at | timestamp
Levels
PK id | integer
key | string
value | string
description | string
created_at | timestamp
updated_at | timestamp

АНАЛІЗ НАЯВНИХ РІШЕНЬ

BITSRENT

StayBit

АНАЛІЗ НАЯВНИХ РІШЕНЬ

Недоліки:

- Тільки короткострокова або довгострокова оренда
- Відсутня гарантія повернення коштів
- Комісійне з’єднання
- Інтерфейс та процес користування

УМОВИ СТВОРЕННЯ СМАРТ-КОНТРАКТУ

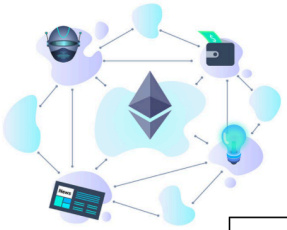
Для створення смарт-контракту потрібно:

Предмет договору

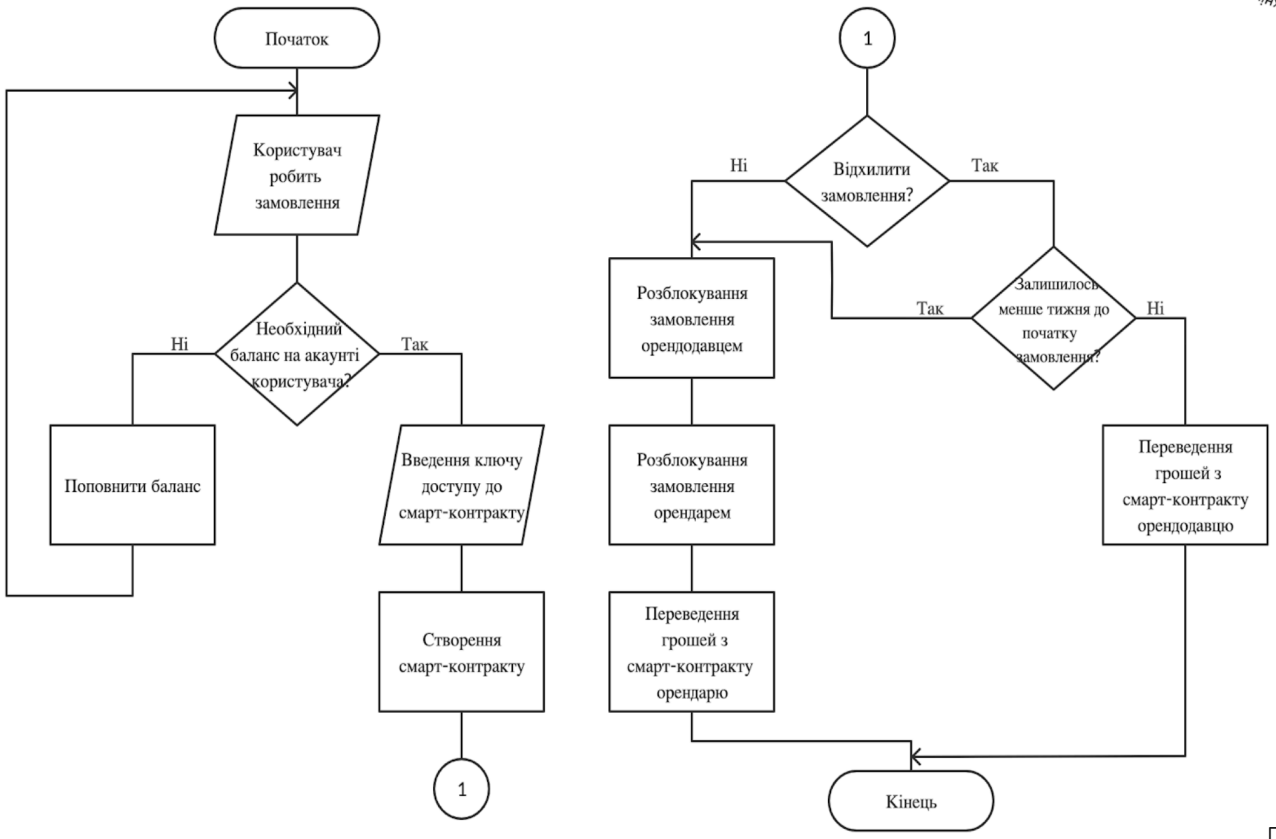
Умови договору

Цифрові підписи

Децентралізована платформа

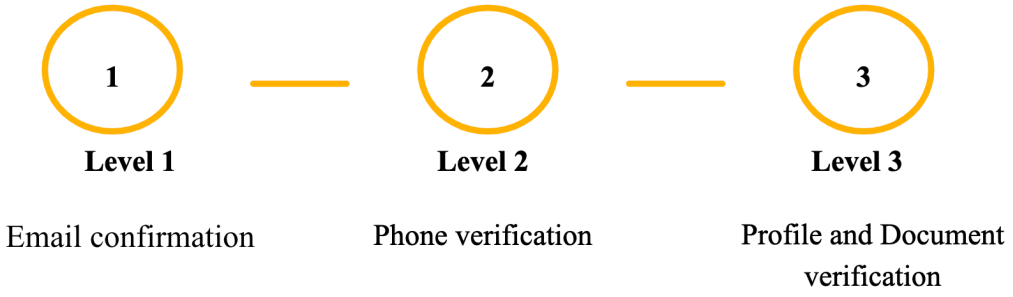


УМОВИ СТВОРЕННЯ СМАРТ-КОНТРАКТУ

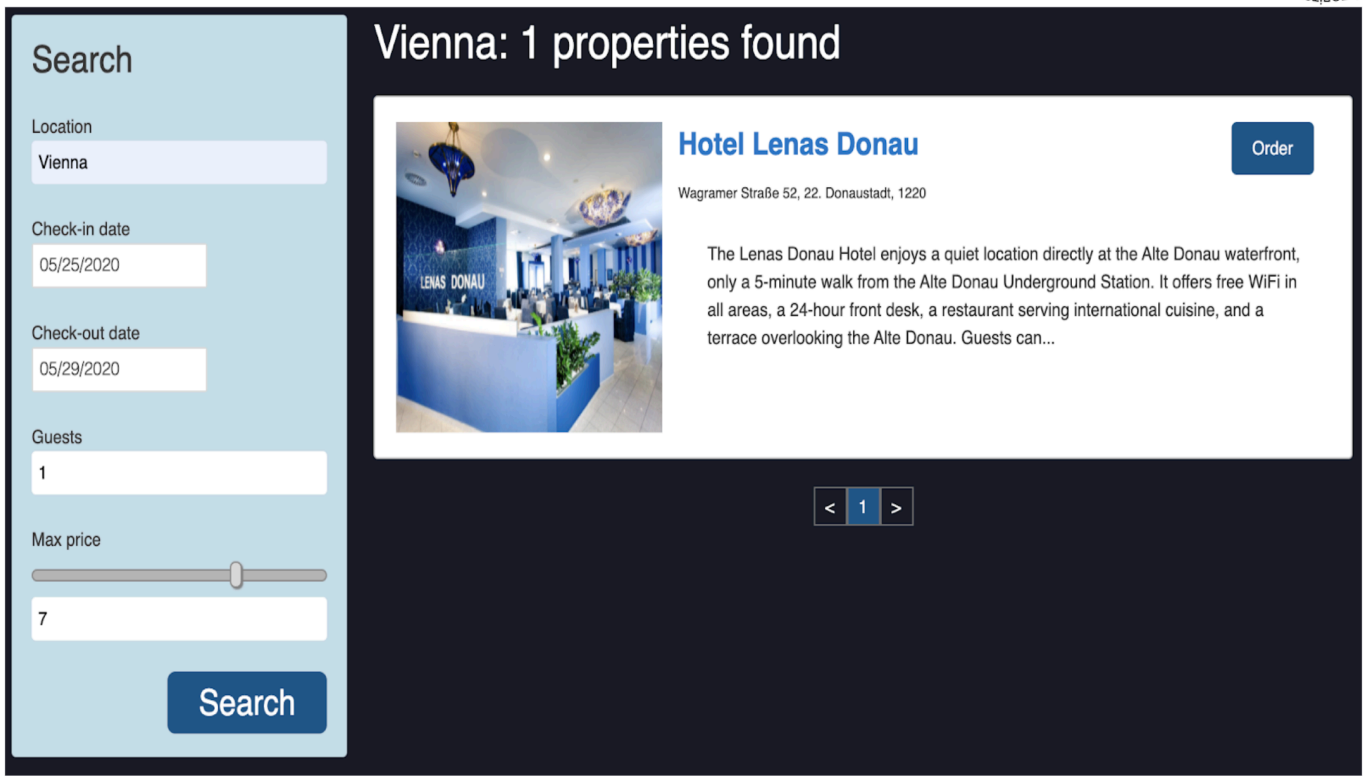


РОЛІ КОРИСТУВАЧІВ В СИСТЕМІ

Адміністратор	—	Level n
Орендар	—	Level 3
Звичайний користувач	—	Level 1

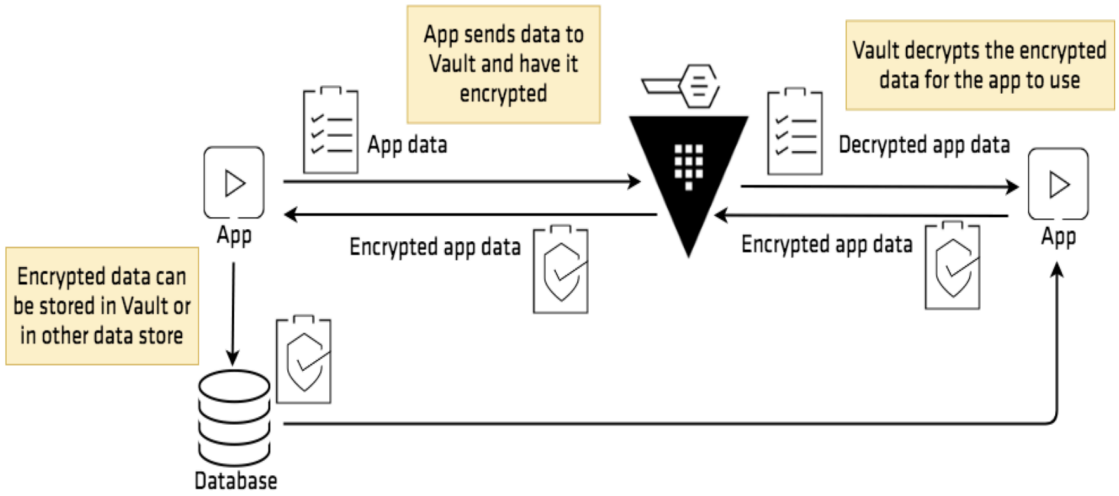


ПРИКЛАД БРОНЮВАННЯ ЗАМОВЛЕННЯ



БЕЗПЕКА ВЕБ-СЕРВІСУ

1. Застосування CSRF токенів для POST запитів
2. Застосування 2FA (Two-factor authentication)
3. Розподілення користувачів на ролі
4. Застосування Vault сервісу для збереження ключів смарт-контрактів



ПОРІВНЯННЯ З АНАЛОГАМИ

	Безкомісійне з’єднання	Оренда будь-якої тривалості	Зрозумілий інтерфейс на процес користування	Захищеність та гарантія повернення коштів
Rentberry	+/–	+/–	+/–	+/–
Bitsrent	+	–	+/–	–
Staybit	+	+	+	+/–
Розроблене ПЗ	+	+	+	+

ШЛЯХИ ПОДАЛЬШОГО РОЗВИТКУ ПРОЄКТУ

- Найм адміністратора на часткову зайнятість для вирішення виняткових ситуацій
- Монетизація трафіку за рахунок реклами
- Створення власного токену
- Залучення інвестицій (проведення ICO)
- Створення модулю статистики для орендодавців

ВИСНОВКИ

1. Проаналізовані існуючі програмні рішення систем укладання договорів оренди
2. Запропоновано мікросервісну архітектуру ПЗ, яка використовує технологію блокчейн та смарт-контракти
3. Розроблено алгоритм смарт-контракту, що включає в себе укладання, контроль виконання договорів оренди та дотримання обов’язків сторін
4. Розроблено ПЗ, яке надає можливість укладання договорів оренди без посередників на основі технології блокчейн
5. Протестовано та порівняно ПЗ з аналогами за такими критеріями: доступна тривалість оренди, наявність комісій за використання, гарантія повернення коштів, досвід користування (UX)

Веб-сервіс реалізовано згідно запланованих вимог (функціональних і нефункціональних), тестування проведено відповідно до затвердженої технології тестування, проведено аналіз відповідності виконаної роботи і поставленого технічного завдання.

ПЕРЕВІРКА НА УНІКАЛЬНІСТЬ

Submission author:
Гадиняк Руслан Анатолійович

Check ID:
1003888624

Check date:
08.06.2020 23:57:12 EEST

Check type:
Doc vs Internet + Library

Report date:
09.06.2020 00:08:33 EEST

User ID:
100000126

File name: **Надія_Чумак_ПЗ**

File ID: **1003902575** Page count: **46** Word count: **9615** Character count: **72130** File size: **324.81 KB**

0.37% Matches

Highest match: **0.12%** with library source. File ID: **1000090507**

No Internet Sources Found

0.37% Library matches 6 ..... Page 48

0.54% Quotes

Quotes 2 ..... Page 49

Exclude references is off

0% Exclusions

No exclusions found

Replacement

No replaced characters found

Дякую за увагу!

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2019 р.

ВЕБ-СЕРВІС ДЛЯ УКЛАДАННЯ БЕЗПЕЧНИХ УГОД ОРЕНДИ НА ОСНОВІ ТЕХНОЛОГІЇ BLOCKCHAIN

Програма та методика тестування

ДП.045440-04-51

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Руслан ГАДИНЯК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Надія ЧУМАК

2019

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Веб-сервіс для укладання безпечних угод оренди на основі технології блокчейн. Дана програмна система буде імплементована у вигляді веб-сервісу, що написана веб-фреймворком Ruby on Rails для серверної частини і веб-фреймворком React для написання клієнтської частини.

2. МЕТА ТЕСТУВАННЯ

Метою тестування є перевірка наступних елементів:

1. Доступ до бази даних через основну серверну частину.
2. Робота сервера із модулем, що відповідає за логіку смарт-контракту.
3. Обробка запитів сторонніх користувачів до відкритого API.
4. Робота функціональних елементів сторінок веб-сторінки.
5. Забезпечення достатньої безпеки даних.
6. Відповідність дизайну вимогам програмного забезпечення.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Black Box Testing, який заснований на тому, що тестувальник має доступ до ПЗ тільки через ті ж інтерфейси, що і замовник або користувач, або зовнішні інтерфейси, що дозволяють іншому комп’ютеру або іншому процесу підключитися до системи для тестування. Black Box Testing не вимагає яких-небудь знань про внутрішню роботу додатка. Процес ґрунтується на аналізі ключових аспектів системи.

Використовуються такі методи тестування:

1. Тестування таблиці рішень.
2. Тестування класу еквівалентності.
3. Тестування діаграм зміни станів.
4. Аналіз граничних значень.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Для тестування методом Black Box Testing використовується бібліотека rspec-rails. RSpec – це інструмент BDD (Bihavior-Driven Development), який використовується для вказівки і тестування програм Ruby. Він використовується, перш за все, для визначення і тестування класів і методів, тобто для модульного тестування.

Повний процес тестування проходить у такому порядку:

1. Тестування інтерфейсу при різних роздільних здатностях екрану.
2. Тестування на відповідність функціональним вимогам.
3. Тестування модулів програми за допомогою бібліотеки Rspec.
4. Тестування модулів на основі діаграм станів сутності.
5. Тестування полей вводу на граничні та неможливі значення.
6. Тестування кількості запитів до API.
7. Тестування у різних браузерах: Chrome, Safari, Firefox.
8. Тестування при максимальному навантаженні комп’ютера на оперативну пам’ять та процесор.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

ВЕБ-СЕРВІС ДЛЯ УКЛАДАННЯ БЕЗПЕЧНИХ УГОД ОРЕНДИ НА ОСНОВІ ТЕХНОЛОГІЇ BLOCKCHAIN

Керівництво користувача

ДП.045440-05-34

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Руслан ГАДИНЯК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Надія ЧУМАК

2020

ЗМІСТ

1. ОПИС СТРУКТУРИ ПРОГРАМИ

Система реалізована програмно у вигляді веб-сервісу, тобто такого додатку, який доступний користувачу як веб-сторінка в його веб-переглядачі, при умові наявності доступу до мережі інтернет. Система складається із поєднання статичних сторінок (вміст яких не змінюється) та контенту, що динамічно (без перезавантаження сторінки) оновлюється.

Веб-сервіс використовує англійську мову як основну. Проте архітектура системи дозволяє додавати переклади і в перспективі кількість таких перекладів необмежена. Таке рішення по вибору основної мови зумовлено тим, що сервіс розрахований на резидентів різних країн, а англійська мова носить статус інтернаціональної, що задовольняє вищеописану умову.

У системі існують такі веб-сторінки:

1. Цільова (головна публічна) сторінка.
2. Сторінка авторизації.
3. Сторінка реєстрації.
4. Сторінка відновлення паролю та доступу до акаунту.
5. Сторінка інформації користувача (профіль).
6. Група сторінок проходження KYC.
    6.1. Сторінка підтвердження телефонного номеру.
    6.2. Сторінка підтвердження особистої інформації.
    6.3. Сторінка підтвердження особи (документів).
7. Сторінка підключення двофакторної авторизації.
8. Сторінка зміни паролю.
9. Сторінка пошуку та фільтрації вільних об’єктів бронювання.
10. Сторінка перегляду інформації об’єкту бронювання.
11. Сторінка історії та фільтрації бронювань користувача.
12. Орендодавець: сторінка перегляду об’єктів бронювання.
13. Орендодавець: сторінка створення нових об’єктів бронювання.

14. Орендодавець: сторінка перегляду списку бронювань об’єктів орендодавця.

Кожна сторінка користувацької частини містить навігаційні елементи, що включають в себе корисні посилання, повернення на попередню сторінку тощо. Вигляд та функціональність навігаційного елементу відрізняється для авторизованих та неавторизованих користувачів.

2. ВМІСТ ВЕБ-СТОРІНОК КОРИСТУВАЧА

Цільова сторінка виконує як маркетингові, так і функціональні завдання. Вона містить інформацію про систему та її призначення, лозунг та опис програми. Також містить посилання на пошук об’єктів бронювання, сторінки реєстрації та авторизації, а також контакт-форму для тих користувачів, які хочуть розмістити власну нерухомість для оренди в системі. Деякі елементи даної сторінки зображені на рис. 2.1.

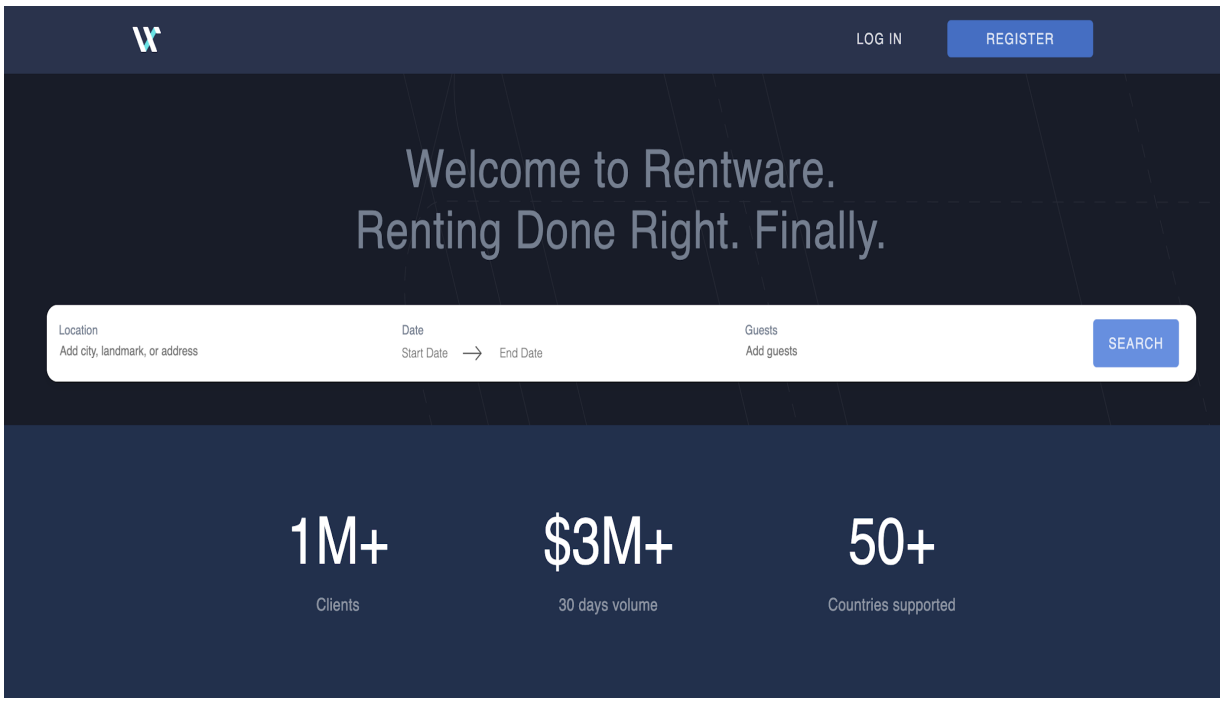


Рис. 2.1. Цільова сторінка

На сторінках реєстрації та авторизації пропонується ввести коректні дані у представлених полях вводу електронної адреси та паролю (рис. 2.2). Кожне поле підсвічується відповідним кольором під час використання, а запит на створення сесії/користувача супроводжується підказкою, що сповіщає користувача про помилку, якщо така є, або про успішне виконання запиту. В разі вводу коректних, правильних даних користувач буде перенаправлений на сторінку пошуку об’єктів оренди бронювання автоматично, без перезавантаження сторінки.

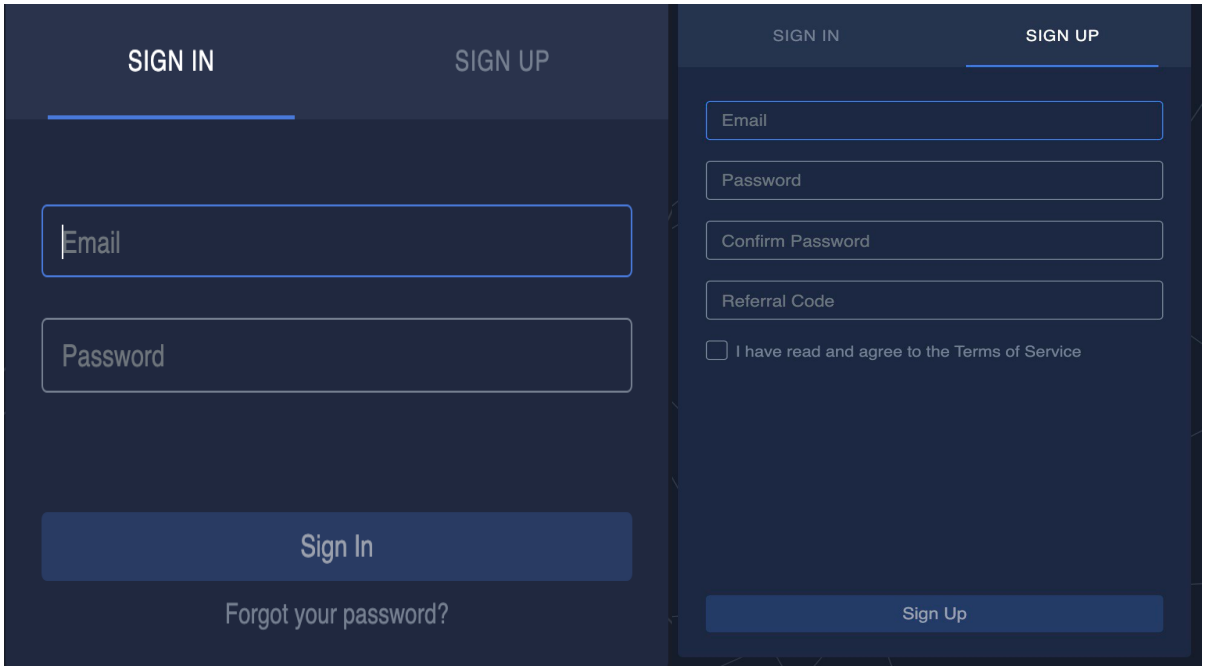


Рис. 2.2. Сторінка реєстрації та авторизації

Сторінка авторизації містить посилання на форму відновлення паролю до акаунту. Посилання міститься під полем введення паролю. Сторінка відновлення пропонує користувачу ввести коректне значення електронної адреси, на яку прийде посилання на сторінку з призначенням нового паролю. Важливо, що сторінка запиту зміни паролю не видає помилки, якщо введена електронна адреса некоректна, тобто не зареєстрована у системі. Це зроблено задля забезпечення безпеки приватності користувачів (рис. 2.3).

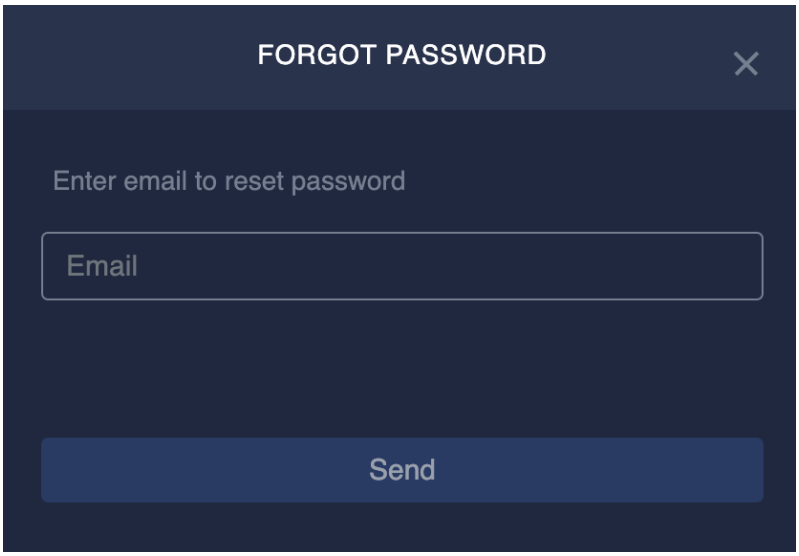


Рис. 2.3. Сторінка відновлення паролю та доступу до акаунту

Будь-який користувач платформи має доступ до власного профілю, де він може переглянути свій поточний рівень КУС, проходження якого повністю обов'язкове лише для орендодавців. Для звичайних користувачів достатньо підтвердити лише електронну адресу. На сторінці свого профілю користувач також має доступ до інформації про поточний баланс та ETH адрес, таблиці активності, де він може переглянути список всіх авторизацій свого акаунту, IP адресу, браузер та час, коли був зроблений запит на авторизацію, тому користувач може періодично перевіряти, чи не було підозрілої активності в його акаунті. Ця інформація надається та зберігається платформою і не може бути змінена користувачем. Сторінка також містить посилання на відповідний етап проходження КУС та сторінку ввімкнення двофакторної авторизації (рис. 2.4). Інтерфейс однаковий для всіх ролей – адміністратора, орендодавця, звичайного користувача. Кожен користувач має ETH адресу, яка генерується під час реєстрації і на яку він повинен перевести відповідні кошти для користування функціональністю платформи.

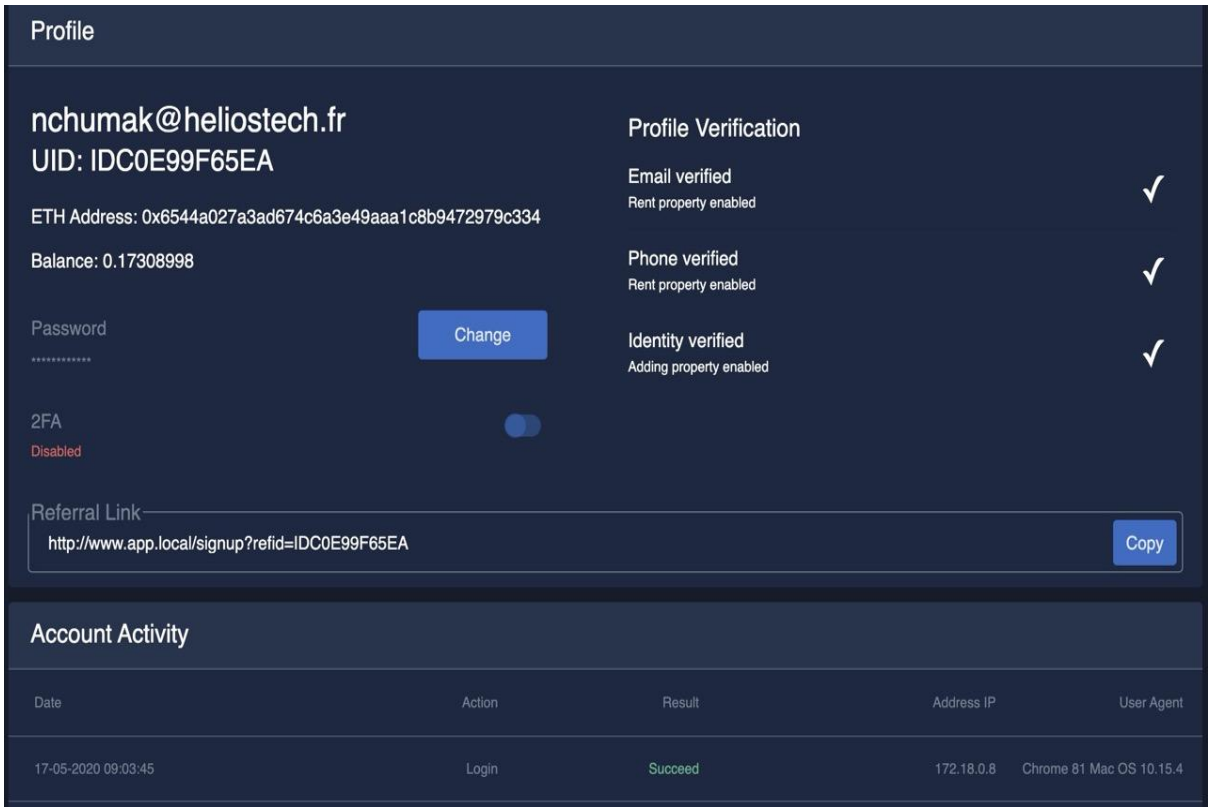


Рис. 2.4. Сторінка інформації користувача (профіль)

KYC (Know Your Customer) – система підтвердження особи, що отримала інтернаціональний статус, в даній програмі реалізована в кілька ступенів, перший з яких – підтвердження електронної адреси, що є обов'язковим пунктом для всіх користувачів. Посилання на наступні рівні міститься на сторінці профілю користувача. Другим рівнем є підтвердження телефону (рис. 2.5). На даній сторінці користувачу пропонується ввести коректний телефонний номер, отримати на нього смс з кодом та ввести код у відповідне поле на сторінці. Кошти за відправку смс не знімаються з балансу користувача, витрати бере на себе платформа. Телефонний номер може бути з будь-якої країни світу, всі вони підтримуються Twilio-сервісом, який використовується у дипломному проєкті. Якщо смс не надійшло, користувач може натиснути кнопку "надіслати знову", при цьому повторно вводити телефонний номер не потрібно. Телефонний номер, закріплений за обліковим записом, повинен бути унікальним.

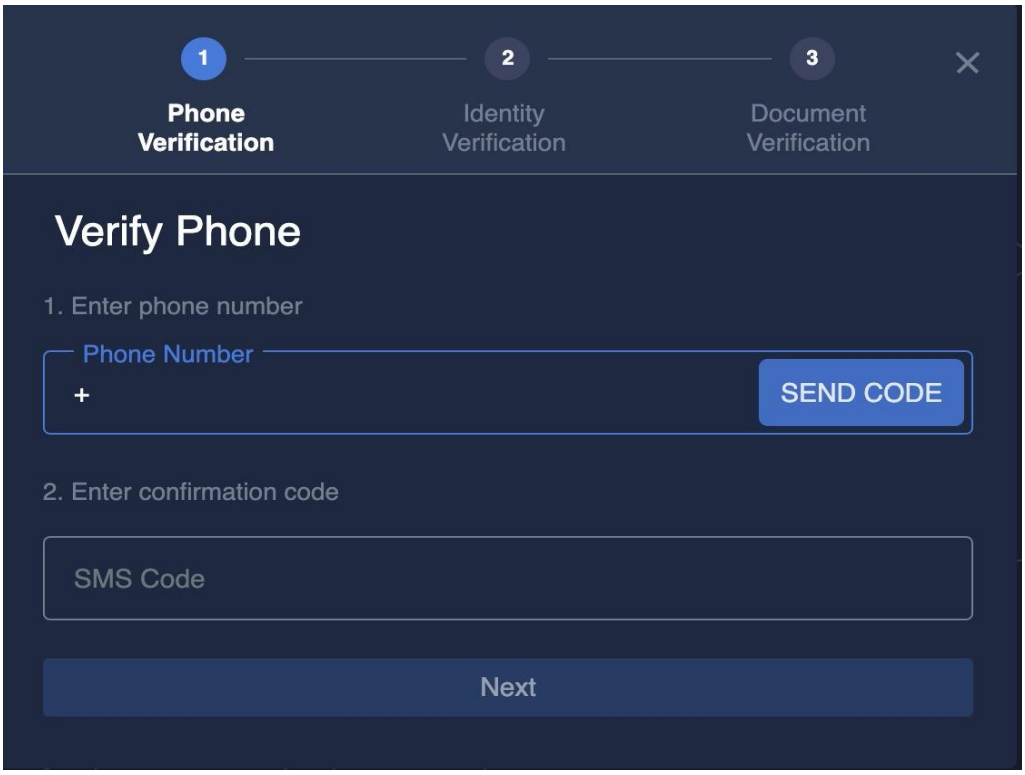


Рис. 2.5. Сторінка підтвердження телефонного номеру

В разі вводу коректного коду номер телефону буде вважатись прив’язаним до даного акаунту, а користувач буде перенаправлений на наступний рівень – сторінку введення особистої інформації. На цій сторінці користувачу пропонується ввести коректні дані в відповідні поля: ім’я, прізвище, дата народження, національність, резидентом якої країни є користувач, адреса резидентства (рис. 2.6). В разі використання не відповідних або заборонених символів поле буде підсвічене червоним кольором. Дата народження повинна бути в форматі число-місяць-рік, що відповідає загальноєвропейському стандарту. Відповідно, кожна частина дати має валідацію на максимальне значення – не більше 31 для числа, не більше 12 для місяця, та не більше ніж поточний рік для останньої частини. Національність та країна проживання можуть відрізнятись одна від одної. Відсутнє поле середнього ім’я, для верифікації достатньо першого та родового.

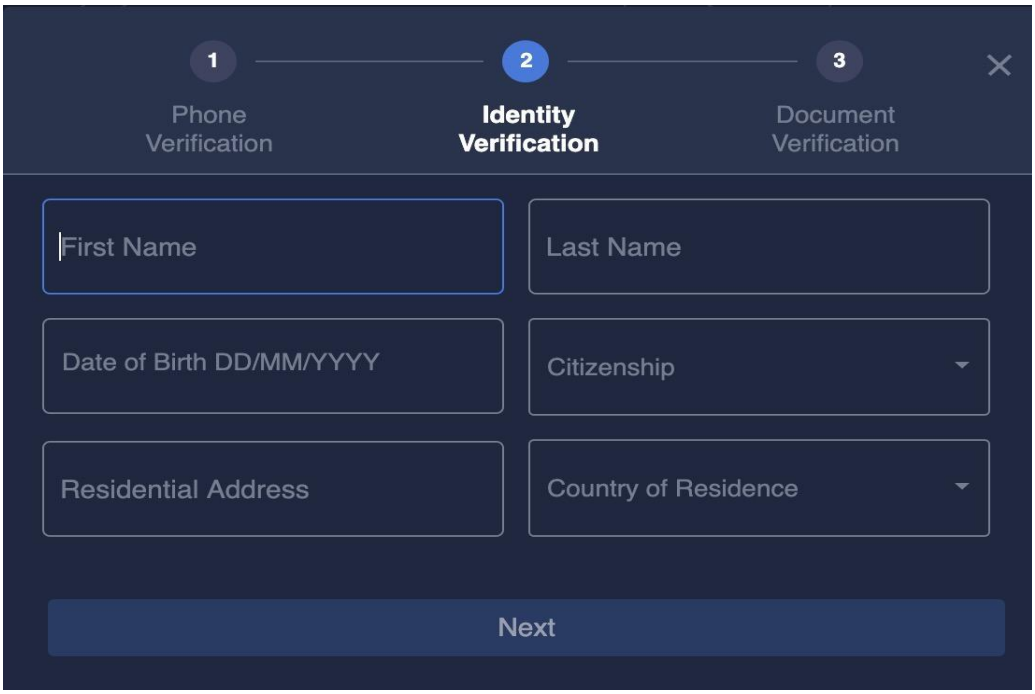


Рис. 2.6. Сторінка підтвердження особистої інформації

Після введення всіх даних та підтвердження форми користувач буде перенаправлений на наступний рівень – сторінку підтвердження особи (рис. 2.7). Дана сторінка містить декілька полів для вводу та поле завантаження документів. Документ повинен бути у зазначеному в формі форматі (png, jpeg, jpg, pdf) та не більше 10MB. Поля вводу обов'язкові та мають певні валідації (поле дати строку придатності документів не повинно бути в минулому тощо), про які користувач буде попереджений в разі вводу некоректних даних. Після підтвердження форми користувач буде перенаправлений на сторінку профілю, а останній рівень (підтвердження особи) буде підсвічений, як "очікує підтвердження". Протягом 2-3 робочих днів адміністрація сервісу перевірить введені дані, відповідність документу та особистої інформації і підтвердить або відмовить у підтвердженні особи. Статус підтвердження користувач побачить на сторінці профілю. В разі відмови, користувач може повторно завантажити документи і отримати нове рішення по підтвердженню особи.

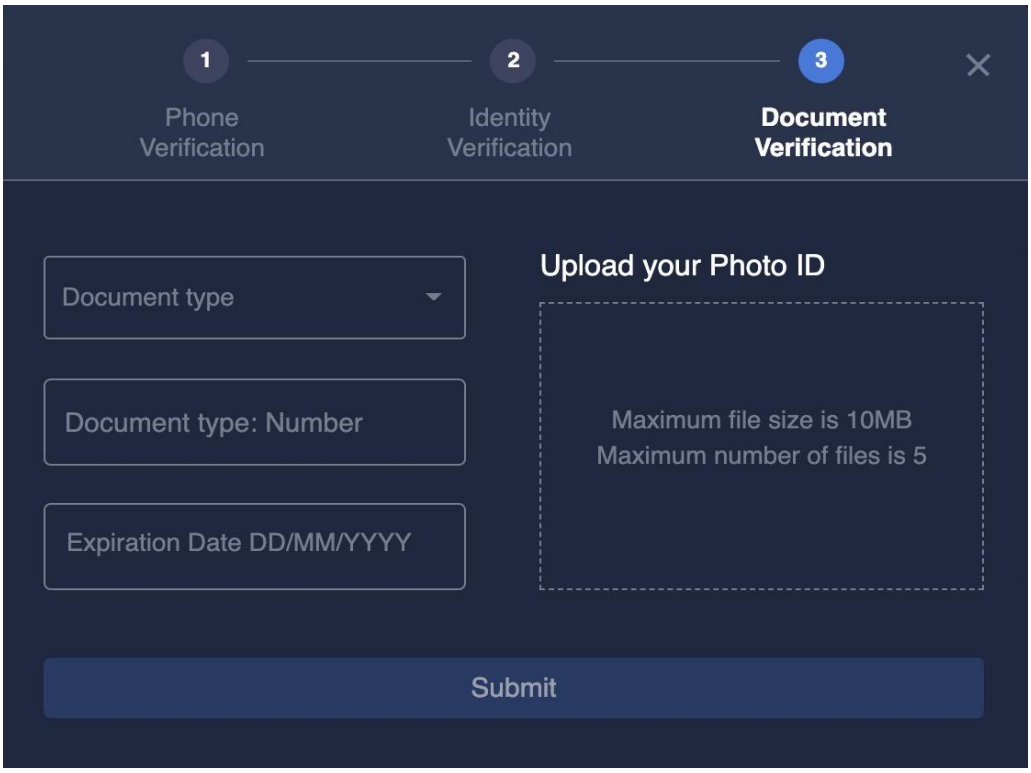


Рис. 2.7. Сторінка підтвердження особи (документів)

Сторінка профілю користувача, як вже було зазначено вище, містить посилання на веб-сторінку ввімкнення двофакторної авторизації (рис. 2.8). Така міра безпеки рекомендована для всіх користувачів, але не є обов'язковою. Дана сторінка містить QR-код та секретний код. Користувач повинен відсканувати QR або ввести секретний код у відповідній мобільній програмі Google authenticator. Після цього в мобільній програмі буде показаний шестизначний код, що оновлюється кожні 30 секунд. Даний код потрібно ввести у відповідне поле на сторінці, щоб підтвердити ввімкнення двофакторної авторизації. З цього моменту кожна нова авторизація потребуватиме введення коду. Якщо людина, яка намагається отримати доступ до облікового запису, вводить неправильний код двофакторної авторизації, то відповідна інформація про неуспішну авторизацію буде збережена у активності акаунту, яку можна переглянути на сторінці профілю користувача.

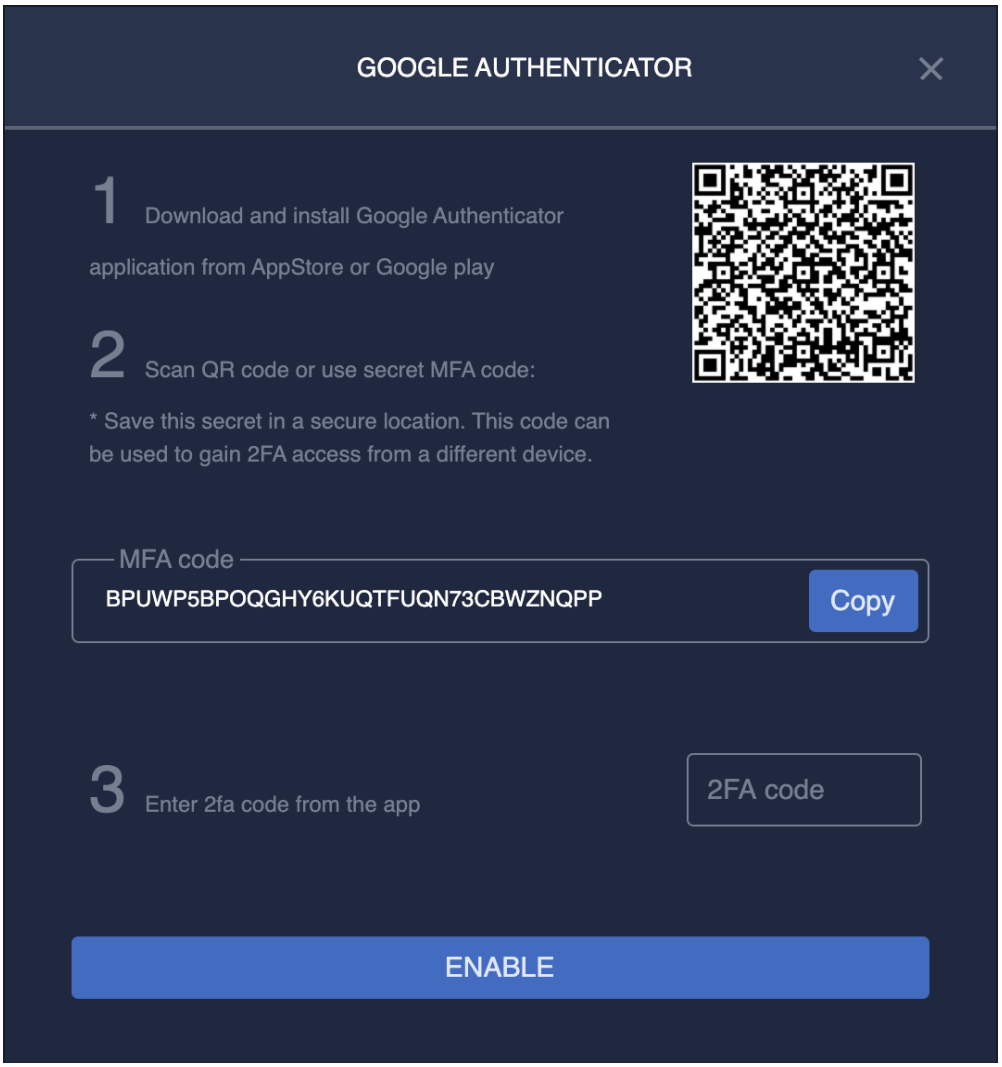


Рис. 2.8. Сторінка підключення двофакторної авторизації

Сторінку профілю містить посилання на сторінку зміни паролю. На даній веб-сторінці користувачу пропонують ввести коректний пароль, що використовується на даний момент, а також двічі ввести нове бажане значення, яке також повинно бути коректним, не містити заборонених символів, а також відповідати стандартам рівню складності, зазначених платформою (рис. 2.9). Після зміни паролю облікового запису буде відправлено лист на адресу акаунту із сповіщенням про факт зміни. Відсутня валідація на ідентичність, якщо користувач введе новий пароль аналогічний старому. В такому випадку з’явиться сповіщення про успішну зміну паролю.

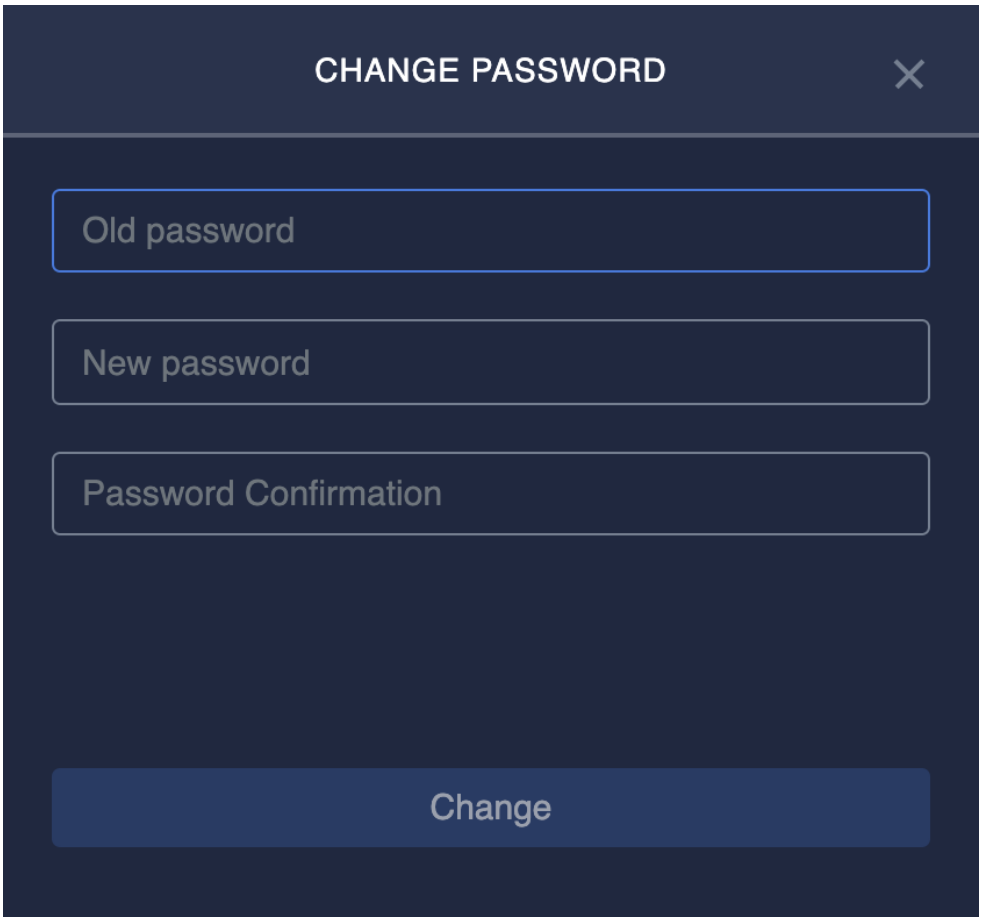


Рис. 2.9. Сторінка зміни паролю

Найбільш функціональною сторінкою для користувача є сторінка пошуку, перегляду і фільтрації списку існуючих та вільних об’єктів бронювання (рис. 2.10). На даній сторінці користувач може переглянути основне фото, задане власником, та короткий опис об’єкту бронювання. Відфільтрувати об’єкти користувач може за такими параметрами (доступними в меню пошуку) – максимальна ціна, місце розміщення об’єкту бронювання, кількість осіб для розміщення та початок-кінець запланованої оренди. Дана сторінка доступна і для авторизованих, і для неавторизованих користувачів. Для переходу на сторінку детальної інформації користувач повинен натиснути на кнопку “Order”. Якщо за заданими параметрами знайдено багато результатів, користувач може переглядати різні сторінками з результатом пошуку.

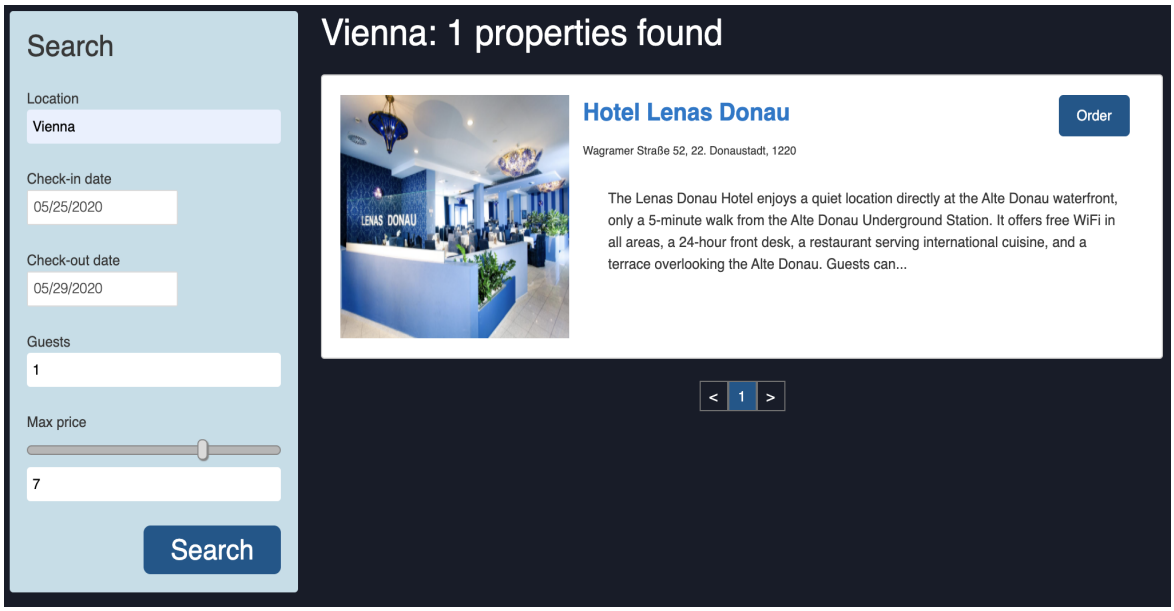


Рис. 2.10. Сторінка пошуку та фільтрації вільних об’єктів бронювання

Кожен об’єкт бронювання має сторінку детальної інформації (рис. 2.11). Тут користувачу надають можливість переглянути всі медіа-матеріали, пов’язані з об’єктом бронювання, детальний опис, представлений власником, а також перелік кімнат, їх ціна за ніч, місткість, назва, короткий опис тощо. Ця сторінка теж публічна, тобто така, яка доступна не авторизованим користувачам. Але вони не мають відповідної кнопки “Reserve”, доступної навпроти кожної з наявних у об’єкта бронювання кімнат. При натисненні на дану кнопку, авторизований користувач, в свою чергу, побачить модальне вікно для введення паролю від бронювання. Паролем, зазвичай, є номер документу, що посвідчує особу, який користувач повинен буде пред’явити під час фактичного заїзду. Якщо в користувача вистачає коштів на акаунті платформи запит на бронювання буде створений, а кошти переведені на адресу смарт-контракту, який буде закріплений за даним бронюванням, і буде його контролювати. Після успішного підтвердження бронювання відповідний запис з’явиться у списку активних бронювань. Бронювання матиме статус “pending” (заплановане) та автоматично змінить статус під час заселення.

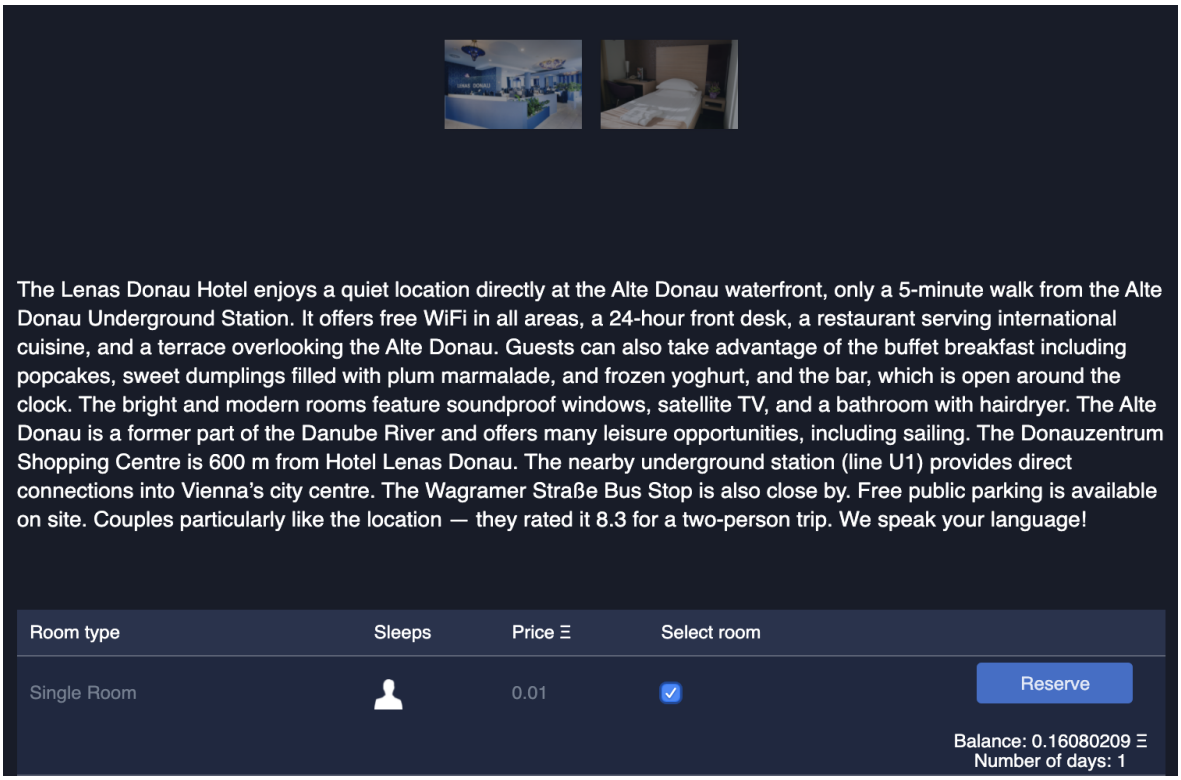


Рис. 2.11. Сторінка перегляду інформації об'єкту бронювання

Створені бронювання, як активні, так і вже закриті, користувач може переглянути на сторінці історії. Посилання на неї міститься в навігаційному елементі і доступне лише для авторизованих користувачів. Історія бронювань доступна в двох форматах, лише активні бронювання (ті, які ще не почались та ті, які проходять в даний момент). Активні бронювання мають додаткову функціональність – ті, що проходять в даний момент мають функцію підтвердження виселення, а заплановані можуть бути відмінені, при умові відміни не менш ніж за 7 днів (рис. 2.12).

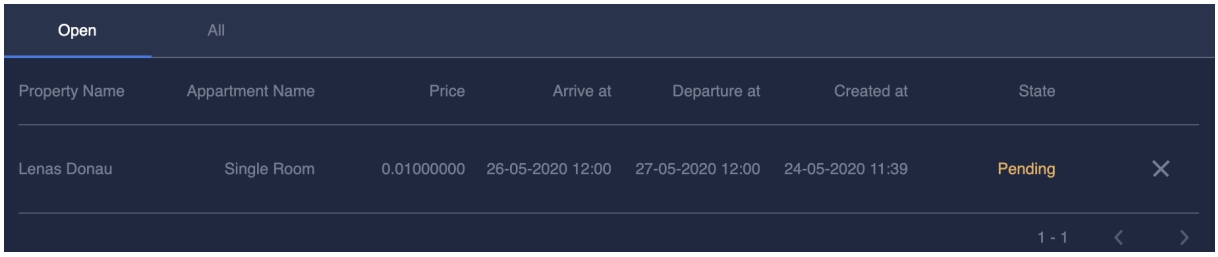


Рис. 2.12. Сторінка активних бронювань користувача

3. ВМІСТ ВЕБ-СТОРІНОК ОРЕНДОДАВЦЯ

Окремим компонентом, відповідальним за функціональність є інтерфейс орендодавця (власника об’єкту бронювання). Дизайн виконано в мінімалістичному стилі, так як основний акцент саме на функціональності та інформативності панелі. Найбільш узагальнений контент, а саме перелік вже існуючих об’єктів бронювань, власником яких є авторизований користувач, є сторінка списку об’єктів бронювань (рис. 3.1). У формі таблиці тут відображені основні елементи кожного з об’єктів – порядковий номер у базі даних, назва, тип, місто, адреса, ETH адреса електронного гаманця, короткий опис та окреме посилання на головну фотокартку об’єкту бронювання.

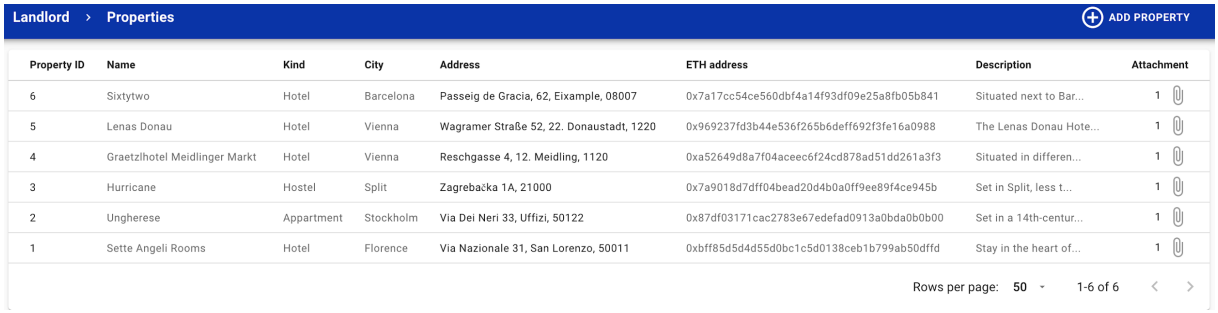


Рис. 3.1. Сторінка перегляду об’єктів бронювання

На сторінці зі списком об’єктів бронювання є посилання на створення нового об’єкту, у правому верхньому куті. Веб-сторінка додавання нового об’єкту бронювання пропонує користувачу ввести відповідні дані у представлені поля для вводу – назва, тип, місто, адреса, ETH адреса гаманця, короткий опис об’єкту та форма завантаження медіа-матеріалу, який буде закріплений за об’єктом бронювання, як головна фотокартка (рис. 3.2).

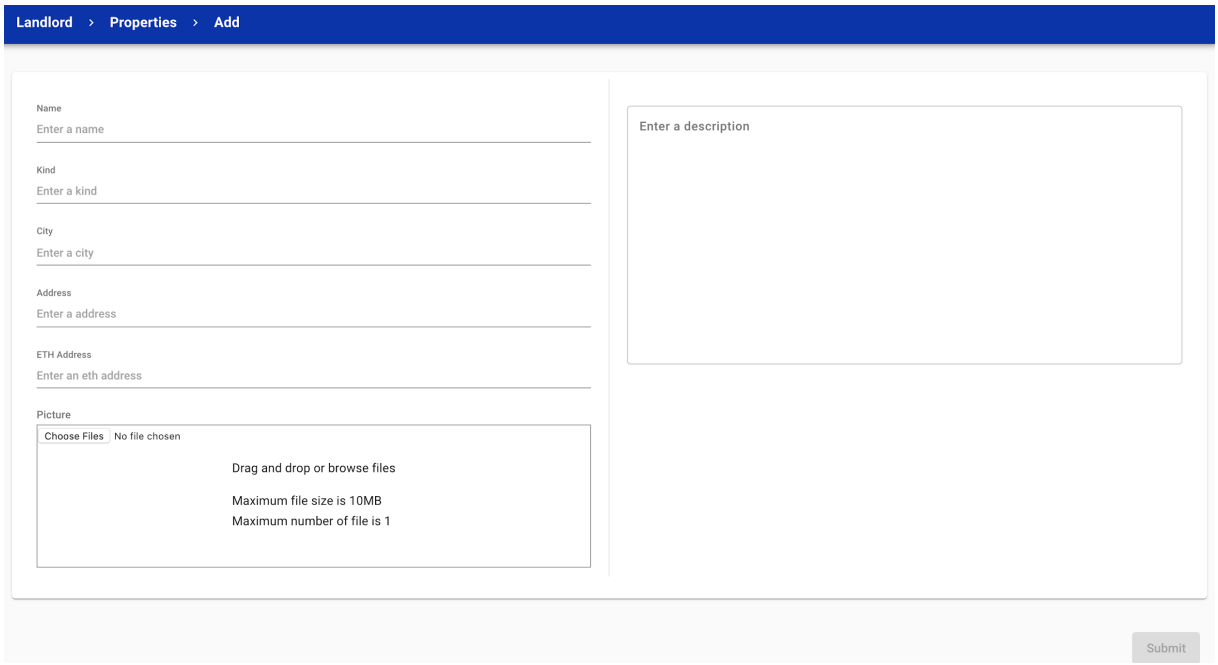


Рис. 3.2. Сторінка створення нового об’єкту бронювання

Навігаційний елемент зліва також містить посилання на список місць розміщення, реалізований у вигляді таблиці, що містить основні елементи – порядковий номер у базі даних, назва об’єкту бронювання, за яким закріплене місце розміщення, назва місця розміщення, кількість доступних місць, ціна за ніч і посилання на медіа-файли об’єкту розміщення (рис. 3.3).

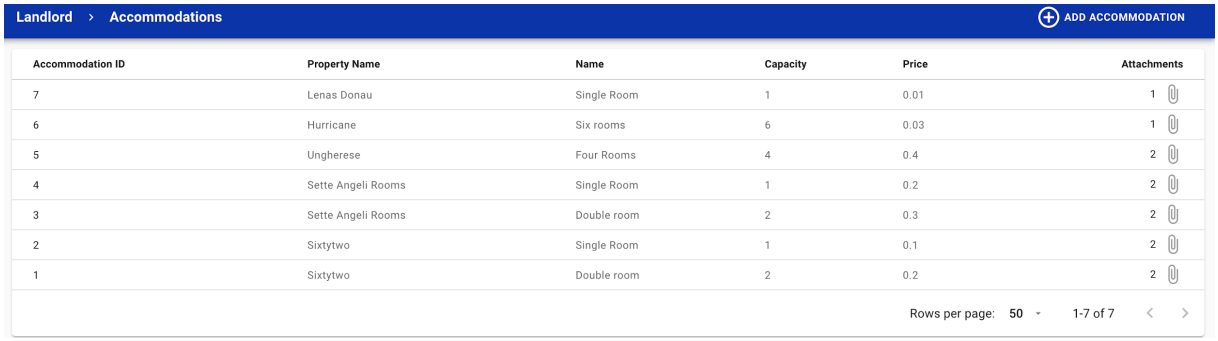


Рис. 3.3. Сторінка перегляду об’єктів розміщення

На сторінці зі списком місць розміщення є посилання на створення нового місця, у правому верхньому куті. Веб-сторінка додавання нового місця розміщення пропонує користувачу ввести відповідні дані у представлені поля для вводу – назва, тип, кількість місць для розміщення,

прив’язка до об’єкту бронювання, ціна за ніч та форма завантаження медіафайлів (рис. 3.4). Після підтвердження користувач буде перенаправлений на сторінку списку місць розміщення.

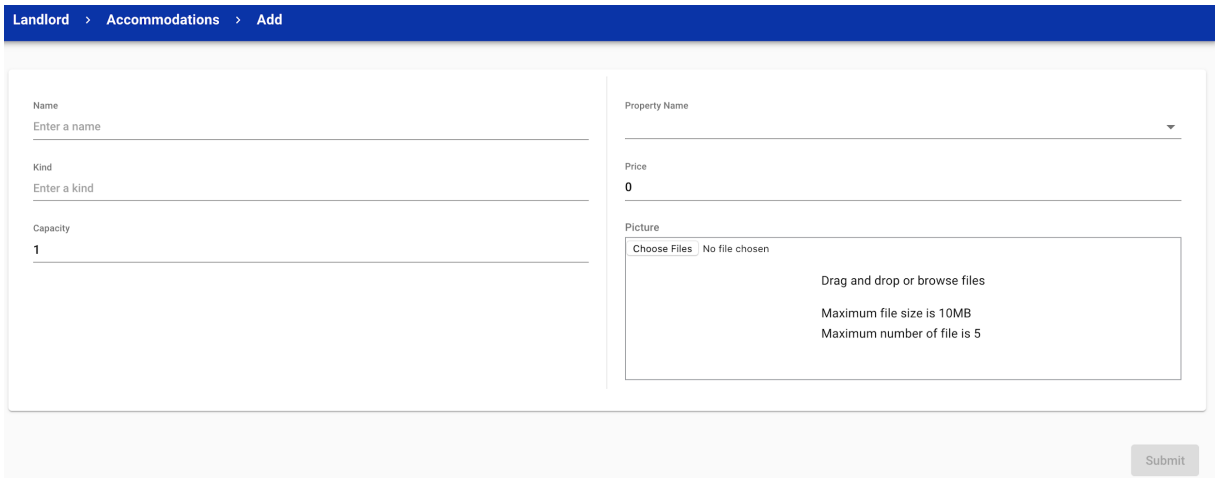


Рис. 3.4. Сторінка створення нового об’єкту розміщення